

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Appl. No.	:	10/658,246	Confirmation No.	4125
Applicant	:	Zizzi		
Filed	:	09/08/2003		
TC/A.U.	:	2132		
Examiner	:	Darrow		
Docket No.	:	M000-P03098US		
Customer No.	:	33356		

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Declaration of Stephen Zizzi

1. I, Stephen Zizzi, am the named inventor on Application No. 10/658,246 ("the '246 application). The '246 application is a continuation-in-part of Application No. 09/259,991 filed March 1, 1999, now US Patent No. 6,981,141, which is a continuation-in-part of Application No. 09/074,191 filed May 7, 1998, now US Patent No. 6,185,181. My curriculum vitae is attached as Exhibit Z-25.
2. All acts and events described herein took place in the United States. More particularly, nearly all took place in Orange County or Los Angeles County, California.
3. References in this declaration to Exhibits Z-2, Z-11, Z-13, Z-14, Z-16, Z-19, Z-21, Z-22, Z-23 and Z-24 are to the exhibits attached to this declaration. The following exhibits on their face show a date by August 1997 and were in fact in existence by that date: Z-2, Z-11, Z-24. The following exhibits on their face show no dates, but were in existence by late August 1997: Z-13, Z-14, Z-16, Z-21, Z-22, Z-23. Exhibit Z-19 has a copyright date of 1998, but was largely in existence and accurately describes the August 1997 version of IntelliGard E. Note that

Exhibit Z-21 references Microsoft Windows 98, which was formally released in 1998. However, Windows 98 was in beta release in early 1997. This is corroborated in the Mahne Dec. ¶ 3.

4. Attached hereto as Exhibit Z-27 is a flow chart of the Microsoft Word event integration process used in IntelliGard E in August, 1997, and this flow chart was recreated based upon my memory and other materials. It is an accurate reproduction of the VBA based integrated of PC DOCS to IntelliGard E in August 1997. This flow chart is corroborated in the Mahne Dec. ¶ 5.
5. Attached hereto as Exhibit Z-28 are excerpts from the annual report of PC DOCS International, Inc. for the year 1997. Though published in 1998, it is retrospective and describes that status of the relevant business and products during 1997. Notably, the auditor's reports at pg. 23 and 47 in Exhibit Z-32 were signed July 29, 1997, and indicate that the financial reporting was for the year ending June 30, 1997. All statements in Exhibit Z-28 referred to herein were true in August, 1997.
6. Attached hereto as Exhibit Z-29 is the source code for the MAZ key server used in IntelliGard E in August, 1997. References to this source code are first to the module name, which appears at the top of each page, and then to the page number within the module. Within each module, the pages are separately numbered. This format also applies to other source code in the exhibits, including Z-2, Z-11, Z-13, Z-16, Z-23 and Z-24.
7. In this declaration, I intend to show that by late August 1997, I had reduced the inventions of claims 13, 19, 29, 31-35, 37-38, 40-42, 44 and 45 to practice, disclosed the inventions to others, publicly demonstrated the system and (through MAZ) offered the invention for sale. The reduction to practice was in the form of MAZ's IntelliGard E product. The disclosure to others was in the form of technical discussions with at least Chris Mahne ("Mahne") and Paul Halpern ("Halpern"), and demonstrations of IntelliGard E to them. *See* Halpern Dec. ¶¶ 4-5, 9-14; Mahne Dec. ¶¶ 13, 23.

8. By late August 1997 MAZ had a version of IntelliGard E that integrated with the PC DOCS EDMS. *See* Halpern Dec. ¶ 11. *See also* Exhibit Z-21, pg. 3, which stated that “DOCS Open 3.7.x” was required to use IntelliGard E. DOCS Open was an off-the-shelf version of the PCDOCS EDMS. An EDMS (electronic document management system) is a “means for businesses to effectively organize their information. Document management has become a vital ingredient in a network user’s ability to easily and quickly locate information, control access to files, track versions of documents, and organize information into project folders.” Exhibit Z-28, pg. 6. “[D]ocument management provides a means to store, easily locate, and retrieve document-based information throughout the document’s life cycle. In other words, the management and control that a document management system provides transforms unorganized document collections into accessible information assets.” Exhibit Z-28, pg. 6.
9. Also by August 1997, MAZ had demonstrated IntelliGard E at Lyon & Lyon, and set pricing for a sale of IntelliGard E to Lyon & Lyon at \$50/seat. *See* Halpern Dec. ¶¶ 11-12; Mahne Dec. ¶ 37.
10. IntelliGard E operated as a plug-in to the PC DOCS EDMS (Electronic Document Management System) through custom designed relational database tables and programming hooks in the form of an events application program interface (API). *See* Mahne Dec. ¶ 2.
11. In addition, a smart card reader was connected to or installed in the PC. *See* Exhibit Z-21, pg. 2 (“A smart card reader comes with IntelliGard System that can easily be connected to the user’s PC”). *See also* Mahne Dec. ¶ 39. The smart card reader interface was developed for several reader devices and referred to as the “keyserver” or “key server”. Source code for this server concept is found in Exhibit Z-29 and Z-16. The smart card stored encryption keys used by IntelliGard E. *See* Exhibit Z-21, pg. 2 (“The encryption key comes from a secure; credit card size storage device called a smart card.”). *See also* Mahne Dec. ¶¶ 23, 39b; Halpern Dec. ¶ 11.

12. Smart cards could be inserted into the reader, and the smart cards were sectioned into a memory stack area that was used as a key table. The key table had key names and corresponding encryption/decryption key values, used in a private (symmetric) key encryption such as DES. *See* Mahne Dec. ¶ 4. Administration and the formatting of this key table design can be found in the sample code in Exhibits Z-11.

13. The following claim chart shows how IntelliGard E had all of the features in independent claims 29, 33 and 40:

Claim Language	Corroboration
Claim 29: A process of decrypting documents comprising:	IntelliGard E was used for decrypting electronic documents. “IntelliGard is a Windows based product that provides file level security for PC users.” Exhibit Z-21, pg. 2. IntelliGard E integrated with PC DOCS, an off-the-shelf electronic document management system. “The IntelliGard system integration with DOCS Open 3.7.x requires a table in the SQL database(s) The DOCS Open Designer Utility is used to create the new database table in SQL.” Exhibit Z-21, pg. 12 (at top).
providing plural documents having respective names	The PC DOCS EDMS included information on all the documents in the PC DOCS EDMS. The PC DOCS EDMS is described in Exhibit Z-28. <i>See e.g.</i> , pg. 6-7. The PC DOCS repository maintained file names, revision levels, author’s name and the security status in an RDBS such as Oracle 7 or MS SQL 6. A description of the table design and data elements is found in Exhibit Z-24.
providing a crypto server for causing documents to be decrypted	IntelliGard E is the claimed “crypto server.”
providing a first table having the names of encrypted documents for each of the names of	IntelliGard E required a table, called the MAZTable, in the PC DOCS SQL database. <i>See</i> Exhibit Z-21, pg. 12-14, which shows how to create the MAZTable. <i>See also</i> Exhibit Z-22, pg. 3 (“after you

<p>encrypted documents in the first table, a key name associated with a decryption key value for the encrypted document</p>	<p>have completed steps I. and II. you must build a New Table in the SQL database used with DOCS Open. Call this table MAZTABLE (very important.”). The MAZTable corresponded to the claimed “first table”.</p> <p>The layout of the MAZTable is shown in Exhibit Z-21, pg. 14. The layout of PC DOCS tables is described in Exhibit Z-24, pg. 3-8 to 3-12. The MAZTable had three columns – DOCNUMBER, SECURE and KEYID. The DOCNUMBER column was an integer field type which provided a unique identifier (i.e., a document name) for documents in the PC DOCS EDMS. The DOCNUMBER was set up as a candidate referential key (see Exhibit Z-24, pg. 3-9, under “Key”) to the DOCNUMBER column in the PC DOCS document profile table and provided reference to the names of the encrypted documents. The KEYID column stored the key names associated with the decryption key values for the encrypted documents. The SECURE column identified whether the document was secure (i.e., encrypted), and had a value of either 1 (i.e., encrypted) or 0 (i.e., unencrypted). An administrator could set SECURE flag by default so that all new documents would be automatically encrypted on close, save or save. The KEYID column was a 33 character string that identified the decryption key for the document (i.e., the key name).</p> <p>The key name was used to retrieve the decryption key value by making a call to the IntelliGard E keyserver. The keyserver was a part of IntelliGard E and was installed with the IntelliGard E main console on the user’s Windows PC. At Windows start up, IntelliGard E would execute and start the keyserver along with other services. Exhibit Z-29, section “clsKeyserver”, pg. 1 shows a Public</p>
---	--

	<p>Property “Get KeyValue” that accepts the argument “sName”. “sName” is the key name found in the KEYID column.</p> <p>The “Get KeyValue” property called a Public Function in the keyserver “ReadKeyValue” (Exhibit Z-29, section “modCardAPI”, pg. 7) that would check to see that it was a valid keyname, find the corresponding index number of the memory location of the key value on the smart card and make a call to the smart card reader using the Public Function “ReadCard” (Exhibit Z-29, section “modCardAPI”, pg. 7). After the key value was fetched from the smart card, the “Get KeyValue” property was set to the decryption key numerical value.</p>
<p>detecting an open command for a given document issuing from a user of an application program using a user input device</p>	<p>IntelliGard E in August 1997 automatically decrypted documents when the documents were opened. This is shown in the flow chart of Exhibit Z-27, which shows that on file open, IntelliGard E trapped the “open” event. Using the MAZTable, IntelliGard E determined if the document was encrypted, and if so, the name of the key to use to decrypt the document. This is shown in the flow chart in Exhibit Z-27, pg. 2.</p> <p>IntelliGard E retrieved the key from the smart card, decrypted the document and then pass control to the PCDOCS EDMS. This is also shown in the flow chart in Exhibit Z-27.</p>
<p>in response to the open command, the crypto server using the first table to determine if the given document should be decrypted</p>	<p>IntelliGard E would access the MAZTable to determine if a document should be decrypted utilizing a PC DOCS API function called “DOCSSQL” (this was also referred to as SQLView). IntelliGard E used the document name (i.e., DOCNUMBER) to check the SECURE flag for the document. The software flow for this is found in Exhibit Z-27, pg. 3 and sample source code for the functions to retrieve data from the MAZTable utilizing the DOCS 16 bit API is found in Exhibit</p>

	<p>Z-23. The first call made to the DOCS API is found in the section “DOCSAPI” on pg. 3. “DOCSBEOpenDocument” determined the fileID of the document currently open in the PC DOCS EDMS (at the time of the transfer of control to IntelliGard E). Using the return value of the fileID, a call to the function “DOCSBeSelectDocument” would set the database record pointer to the proper row in the database. The DOCNUMBER column created in the MAZTable was a “candidate” referential key (see the diagram and column description in Exhibit Z-21, pg. 14). The referential integrity of the database associated the DOCNUMBER to the fileID field that was part of the record where the document name is found. In section “DOCSAPI” pg. 5, the function “DOCSSQLGetColumnValue” was called to get the value for the database column “SECURE”. The return of the function was a 1 if the document record indicated that the document was to be decrypted. If the SECURE field was a binary 1 (yes), then the decryption process event continued. If it was false, the process would open the file in clear text, and exit the VBA code thus returning control to the PC DOCS EDMS (see Exhibit Z-27, pg. 1 flowchart items (k) – (m). <i>See also</i> Halpern Dec. ¶ 14c; Mahne Dec. ¶ 39c.</p>
<p>if the given document should be decrypted, then</p> <p>retrieving the key name associated with the name of the given document from the first table</p>	<p>If the document should be decrypted, IntelliGard E automatically obtained a decryption key value for the document.</p> <p>The MAZTable associated documents names (DOCNUMBER) with key names (KEYID). Using PC DOCS SQLview, IntelliGard E used the DOCNUMBER to obtain the key name (i.e., KEYID) from the MAZTable (the software process for this is found in Exhibit Z-27, pg. 4).</p> <p>To retrieve the key identity (KEYID) the “DOCSSQLGetColumnValue” (as</p>

	described above) would again be called with the column name set to “KEYID” which would return the key name associated with the document. <i>See</i> Halpern Dec. ¶ 14c-d; Mahne Dec. ¶ 39c-d.
retrieving the decryption key value associated with the key name from a second table, the second table having at least one decryption key value	<p>As explained above, IntelliGard E worked with smart cards, and the smart cards stored key values in a table, in association with respective key names. The smart card’s key table corresponded to the claimed “second table”. After IntelliGard E obtained the key name from the MAZTable, IntelliGard E would use the KEYID to retrieve the key value from the smart card’s key table. <i>See also</i> Halpern Dec. ¶ 14d; Mahne Dec. ¶ 39d. Sample source code for the reading and administration of the smart card and the smart card’s key table is found in Exhibits Z-11, Z-13, Z-14. In Exhibit Z-11, section “modCardInterface” there are a number of functions that were used to manage the smart card. Public Function “GetUserName” and “GetPassword” on pg. 4 would return the smart card’s security information that protected access to the key table. Also on pg. 4 was the Public Function “GetFileStruct” and on pg. 5 “GetCardType”, “GetAllowedKeyCount” and “GetKeyCount” provided information as to the key table type and size that is found on the inserted smart card. In Exhibit Z-11, section “modMain” the Public Function on pg.1 “WriteKeys” is a sample of how the key names, key values, and the date created were written to the smart card file.</p> <p>Exhibit Z-29 is sample source code for the key server that allowed IntelliGard programs to read from the smart card. This is the clsKeyServer object referred to in Exhibit Z-2, section “Igardenmain”, pg. 1. To retrieve the key value by keyID (name) from a smart card, a call was made to a Property Get “Get KeyValue” as described</p>

	in Exhibit Z-29, section “clsKeyServer”, pg. 1. <i>See also</i> Halpern Dec. ¶ 14d; Mahne Dec. ¶ 39d.
causing the given document to be decrypted.	<p>In August 1997 IntelliGard E used the DES algorithm. <i>See</i> Exhibit Z-21, pg. 2 (“All files are decrypted using the Data Encryption Standard (DES) algorithm and a decryption key”). <i>See also</i> Mahne Dec. ¶ 27.</p> <p>After retrieving the key value from the smart card, IntelliGard E automatically caused the document to be decrypted using the decryption key value. <i>See</i> Halpern Dec. ¶ 14d; Mahne Dec. ¶ 39d. Exhibit Z-27, pg. 5 is a flow chart of the decryption process which shows this step. To decrypt a file a call was made to the decryption engine function in the deceng.dll. (This .dll was installed with the IntelliGard E software on the user’s PC.) The DecFile function made a call to “DecryptFile”. The DecryptFile function arguments included the arguments “KeyID” (keyname) and the “CardKey” (key value) that were passed in at the time of call. DecryptFile decrypted the document using a decryption engine and used the DES algorithm.</p>
Claim 33: A computer program product comprising a computer usable medium having computer readable program code embodied therein for decrypting documents, the program code for causing a processor to	<p>IntelliGard E required a PC running Windows and PC DOCS. <i>See</i> Exhibit Z-21, pg. 3 (“System Requirements . . . IBM compatible 486 PC . . . Microsoft Windows® 95, 98, NT4.0”). The Windows PC was a general purpose computer, and it is well known that application programs such as MS Word ran on Windows PCs and processed electronic documents. These processes included opening, closing, and saving the electronic documents.</p> <p>IntelliGard E was used for decrypting electronic documents. “IntelliGard is a Windows based product that provides file level security for PC users.” Exhibit Z-21, pg. 2. IntelliGard E integrated with PC</p>

	DOCS, an off-the-shelf electronic document management system. “The IntelliGard system integration with DOCS Open 3.7.x requires a table in the SQL database(s) The DOCS Open Designer Utility is used to create the new database table in SQL.” Exhibit Z-21, pg. 12 (at top).
cause plural documents to be decrypted, the documents having respective names	<p>The PC DOCS EDMS included information on all the documents in the PC DOCS EDMS. The PC DOCS EDMS is described in Exhibit Z-28. <i>See e.g.</i>, pg. 6-7. The PC DOCS repository maintained file names, revision levels, author’s name and the security status in an RDBS such as Oracle 7 or MS SQL 6. A description of the table design and data elements is found in Exhibit Z-24.</p> <p>In August 1997 IntelliGard E used the DES algorithm. <i>See</i> Exhibit Z-21, pg. 2 (“All files are decrypted using the Data Encryption Standard (DES) algorithm and a decryption key”). <i>See</i> also Mahne Dec. ¶ 27.</p> <p>After retrieving the key value from the smart card, IntelliGard E automatically caused the document to be decrypted using the decryption key value. <i>See</i> Halpern Dec. ¶ 14d; Mahne Dec. ¶ 39d. Exhibit Z-27, pg. 5 is a flow chart of the decryption process which shows this step. To decrypt a file a call was made to the decryption engine function in the deceng.dll. (This .dll was installed with the IntelliGard E software on the user’s PC.) The DecFile function made a call to “DecryptFile”. The DecryptFile function arguments included the arguments “KeyID” (keyname) and the “CardKey” (key value) that were passed in at the time of call. DecryptFile decrypted the document using a decryption engine and used the DES algorithm.</p>
record in a first table	IntelliGard E required a table, called the MAZTable, in the PC DOCS SQL

<p>the names of the encrypted documents</p> <p>for each of the names of encrypted documents in the first table, a key name associated with a decryption key value for the encrypted document</p>	<p>database. <i>See</i> Exhibit Z-21, pg. 12-14, which shows how to create the MAZTable. <i>See also</i> Exhibit Z-22, pg. 3 (“after you have completed steps I. and II. you must build a New Table in the SQL database used with DOCS Open. Call this table MAZTABLE (very important).”). The MAZTable corresponded to the claimed “first table”.</p> <p>The layout of the MAZTable is shown in Exhibit Z-21, pg. 14. The layout of PC DOCS tables is described in Exhibit Z-24, pg. 3-8 to 3-12. The MAZTable had three columns – DOCNUMBER, SECURE and KEYID. The DOCNUMBER column was an integer field type which provided a unique identifier (i.e., a document name) for documents in the PC DOCS EDMS. The DOCNUMBER was set up as a candidate referential key (see Exhibit Z-24, pg. 3-9, under “Key”) to the DOCNUMBER column in the PC DOCS document profile table and provided reference to the names of the encrypted documents. The KEYID column stored the key names associated with the decryption key values for the encrypted documents. The SECURE column identified whether the document was secure (i.e., encrypted), and had a value of either 1 (i.e., encrypted) or 0 (i.e., unencrypted). An administrator could set SECURE flag by default so that all new documents would be automatically encrypted on close, save or save. The KEYID column was a 33 character string that identified the decryption key for the document (i.e., the key name).</p> <p>The key name was used to retrieve the decryption key value by making a call to the IntelliGard E keyserver. The keyserver was a part of IntelliGard E and was installed with the IntelliGard E main console on the user’s Windows PC. At Windows start up, IntelliGard E would</p>
--	---

	<p>execute and start the keyserver along with other services. Exhibit Z-29, section “clsKeyserver”, pg. 1 shows a Public Property “Get KeyValue” that accepts the argument “sName”. “sName” is the key name found in the KEYID column.</p> <p>The “Get KeyValue” property called a Public Function in the keyserver “ReadKeyValue” (Exhibit Z-29, section “modCardAPI”, pg. 7) that would check to see that it was a valid keyname, find the corresponding index number of the memory location of the key value on the smart card and make a call to the smart card reader using the Public Function “ReadCard” (Exhibit Z-29, section “modCardAPI”, pg. 7). After the key value was fetched from the smart card, the “Get KeyValue” property was set to the decryption key numerical value.</p>
<p>detect an open command for a given document issuing from a user of an application program using a user input device</p>	<p>IntelliGard E in August 1997 automatically decrypted documents when the documents were opened. This is shown in the flow chart of Exhibit Z-27, which shows that on file open, IntelliGard E trapped the “open” event. Using the MAZTable, IntelliGard E determined if the document was encrypted, and if so, the name of the key to use to decrypt the document. This is shown in the flow chart in Exhibit Z-27, pg. 2. IntelliGard E retrieved the key from the smart card, decrypted the document and then pass control to the PCDOCS EDMS. This is also shown in the flow chart in Exhibit Z-27.</p>
<p>in response to the open command use the first table to determine if the given document should be decrypted</p>	<p>IntelliGard E would access the MAZTable to determine if a document should be decrypted utilizing a PC DOCS API function called “DOCSSQL” (this was also referred to as SQLView). IntelliGard E used the document name (i.e., DOCNUMBER) to check the SECURE flag for the document. The software flow for this is found in Exhibit Z-27, pg. 3 and</p>

	<p>sample source code for the functions to retrieve data from the MAZTable utilizing the DOCS 16 bit API is found in Exhibit Z-23. The first call made to the DOCS API is found in the section “DOCSAPI” on pg. 3. “DOCSBEOpenDocument” determined the fileID of the document currently open in the PC DOCS EDMS (at the time of the transfer of control to IntelliGard E). Using the return value of the fileID, a call to the function “DOCSBeSelectDocument” would set the database record pointer to the proper row in the database. The DOCNUMBER column created in the MAZTable was a “candidate” referential key (see the diagram and column description in Exhibit Z-21, pg. 14). The referential integrity of the database associated the DOCNUMBER to the fileID field that was part of the record where the document name is found. In section “DOCSAPI” pg. 5, the function “DOCSSQLGetColumnValue” was called to get the value for the database column “SECURE”. The return of the function was a 1 if the document record indicated that the document was to be decrypted.</p> <p>If the SECURE field was a binary 1 (yes), then the decryption process event continued. If it was false, the process would open the file in clear text, and exit the VBA code thus returning control to the PC DOCS EDMS (see Exhibit Z-27, pg. 1 flowchart items (k) – (m). <i>See also</i> Halpern Dec. ¶ 14c; Mahne Dec. ¶ 39c.</p>
<p>if the given document should be decrypted, then</p> <p>retrieve the key name associated with the name of the given document from the first table</p>	<p>If the document should be decrypted, IntelliGard E automatically obtained a decryption key value for the document.</p> <p>The MAZTable associated documents names (DOCNUMBER) with key names (KEYID). Using PC DOCS SQLview, IntelliGard E used the DOCNUMBER to obtain the key name (i.e., KEYID) from the MAZTable (the software process for this is found in Exhibit Z-27, pg. 4).</p>

	<p>To retrieve the key identity (KEYID) the “DOCSSQLGetColumnValue” (as described above) would again be called with the column name set to “KEYID” which would return the key name associated with the document. <i>See</i> Halpern Dec. ¶ 14c-d; Mahne Dec. ¶ 39c-d.</p>
<p>retrieve the decryption key value associated with the key name from a second table, the second table having at least one decryption key value</p>	<p>As explained above, IntelliGard E worked with smart cards, and the smart cards stored key values in a table, in association with respective key names. The smart card’s key table corresponded to the claimed “second table”. After IntelliGard E obtained the key name from the MAZTable, IntelliGard E would use the KEYID to retrieve the key value from the smart card’s key table. <i>See also</i> Halpern Dec. ¶ 14d; Mahne Dec. ¶ 39d. Sample source code for the reading and administration of the smart card and the smart card’s key table is found in Exhibits Z-11, Z-13, Z-14. In Exhibit Z-11, section “modCardInterface” there are a number of functions that were used to manage the smart card. Public Function “GetUserName” and “GetPassword” on pg. 4 would return the smart card’s security information that protected access to the key table. Also on pg. 4 was the Public Function “GetFileStruct” and on pg. 5 “GetCardType”, “GetAllowedKeyCount” and “GetKeyCount” provided information as to the key table type and size that is found on the inserted smart card. In Exhibit Z-11, section “modMain” the Public Function on pg.1 “WriteKeys” is a sample of how the key names, key values, and the date created were written to the smart card file.</p> <p>Exhibit Z-29 is sample source code for the key server that allowed IntelliGard programs to read from the smart card. This is the clsKeyServer object referred to in Exhibit Z-2, section “Igardenmain”, pg. 1.</p>

	<p>To retrieve the key value by keyID (name) from a smart card, a call was made to a Property Get “Get KeyValue” as described in Exhibit Z-29, section “clsKeyServer”, pg. 1. <i>See also</i> Halpern Dec. ¶ 14d; Mahne Dec. ¶ 39d.</p>
<p>cause the given document to be decrypted.</p>	<p>In August 1997 IntelliGard E used the DES algorithm. <i>See</i> Exhibit Z-21, pg. 2 (“All files are decrypted using the Data Encryption Standard (DES) algorithm and a decryption key”). <i>See also</i> Mahne Dec. ¶ 27.</p> <p>After retrieving the key value from the smart card, IntelliGard E automatically caused the document to be decrypted using the decryption key value. <i>See</i> Halpern Dec. ¶ 14d; Mahne Dec. ¶ 39d. Exhibit Z-27, pg. 5 is a flow chart of the decryption process which shows this step. To decrypt a file a call was made to the decryption engine function in the deceng.dll. (This .dll was installed with the IntelliGard E software on the user’s PC.) The DecFile function made a call to “DecryptFile”. The DecryptFile function arguments included the arguments “KeyID” (keyname) and the “CardKey” (key value) that were passed in at the time of call. DecryptFile decrypted the document using a decryption engine and used the DES algorithm.</p>
<p>Claim 40: A computer program product comprising a computer usable medium having computer readable program code embodied therein for encrypting documents, the program code for causing a processor to</p>	<p>IntelliGard E required a PC running Windows and PC DOCS. <i>See</i> Exhibit Z-21, pg. 3 (“System Requirements . . . IBM compatible 486 PC . . . Microsoft Windows® 95, 98, NT4.0”). The Windows PC was a general purpose computer, and it is well known that application programs such as MS Word ran on Windows PCs and processed electronic documents. These processes included opening, closing, and saving the electronic documents.</p> <p>IntelliGard E was used for encrypting electronic documents. “IntelliGard is a</p>

	<p>Windows based product that provides file level security for PC users.” Exhibit Z-21, pg. 2. IntelliGard E integrated with PC DOCS, an off-the-shelf electronic document management system. “The IntelliGard system integration with DOCS Open 3.7.x requires a table in the SQL database(s). The DOCS Open Designer Utility is used to create the new database table in SQL.” Exhibit Z-21, pg. 12 (at top).</p>
<p>cause plural documents to be encrypted, the documents having respective names</p>	<p>IntelliGard E was used for encrypting electronic documents. “IntelliGard is a Windows based product that provides file level security for PC users.” Exhibit Z-21, pg. 2. IntelliGard E integrated with PC DOCS, an off-the-shelf electronic document management system. “The IntelliGard system integration with DOCS Open 3.7.x requires a table in the SQL database(s) The DOCS Open Designer Utility is used to create the new database table in SQL.” Exhibit Z-21, pg. 12 (at top).</p> <p>The PC DOCS EDMS included information on all the documents in the PC DOCS EDMS. The PC DOCS EDMS is described in Exhibit Z-28. <i>See e.g.</i>, pg. 6-7. The PC DOCS repository maintained file names, revision levels, author’s name and the security status in an RDBS such as Oracle 7 or MS SQL 6. A description of the table design and data elements is found in Exhibit Z-24.</p>
<p>record in a first table</p> <p>the names of the encrypted documents</p> <p>for each of the names of encrypted documents in the first table, a key name associated with an encryption key value for the encrypted document</p>	<p>IntelliGard E required a table, called the MAZTable, in the PC DOCS SQL database. <i>See</i> Exhibit Z-21, pg. 12-14, which shows how to create the MAZTable. <i>See also</i> Exhibit Z-22, pg. 3 (“after you have completed steps I. and II. you must build a New Table in the SQL database used with DOCS Open. Call this table MAZTABLE (very important).”). The MAZTable corresponded to the claimed “first table”.</p>

	<p>The layout of the MAZTable is shown in Exhibit Z-21, pg. 14. The layout of PC DOCS tables is described in Exhibit Z-24, pg. 3-8 to 3-12. The MAZTable had three columns – DOCNUMBER, SECURE and KEYID. The DOCNUMBER column was an integer field type which provided a unique identifier (i.e., a document name) for documents in the PC DOCS EDMS. The DOCNUMBER was set up as a candidate referential key (see Exhibit Z-24, pg. 3-9, under “Key”) to the DOCNUMBER column in the PC DOCS document profile table and provided reference to the names of the encrypted documents. The KEYID column stored the key names associated with the encryption key values for the encrypted documents. The SECURE column identified whether the document was secure (i.e., encrypted), and had a value of either 1 (i.e., encrypted) or 0 (i.e., unencrypted). An administrator could set SECURE flag by default so that all new documents would be automatically encrypted on close, save or save. The KEYID column was a 33 character string that identified the encryption key for the document (i.e., the key name).</p> <p>The key name was used to retrieve the encryption key value by making a call to the IntelliGard E keyserver. The keyserver was a part of IntelliGard E and was installed with the IntelliGard E main console on the user’s Windows PC. At Windows start up, IntelliGard E would execute and start the keyserver along with other services. Exhibit Z-29, section “clsKeyserver”, pg. 1 shows a Public Property “Get KeyValue” that accepts the argument “sName”. “sName” is the key name found in the KEYID column.</p> <p>The “Get KeyValue” property called a Public Function in the keyserver</p>
--	---

	<p>“ReadKeyValue” (Exhibit Z-29, section “modCardAPI”, pg. 7) that would check to see that it was a valid keyname, find the corresponding index number of the memory location of the key value on the smart card and make a call to the smart card reader using the Public Function “ReadCard” (Exhibit Z-29, section “modCardAPI”, pg. 7). After the key value was fetched from the smart card, the “Get KeyValue” property was set to the encryption key numerical value.</p>
<p>detect a close command for a given document issuing from a user of an application program using a user input device</p>	<p>It is well known that Windows PCs included a monitor (display), a keyboard and a mouse as user input devices, and an Intel processor.</p> <p>It was common practice for Windows PC users to issue “close,” “save” or “save as” commands from application programs such as MS Word. This was done using the mouse or keyboard. This is shown in the flow chart in Exhibit Z-27.</p> <p>In the version of IntelliGard E which we installed, demonstrated and offered to sell to Lyon & Lyon by August 1997, we embedded macros in a MS Word document template. The macros created a new set of file “open”, “save” and “save as” command “buttons” in the users application. Clicking on these command keys would call a function written in VBA (Visual Basic for Applications) that would translate the command into programming events. <i>See</i> Exhibit Z-27, pg. 1 flowchart items (a) and (a1), and Mahne Dec. ¶ 5. This is also described in Exhibit Z-19, pp. 7-8.</p>
<p>in response to the close command use the first table to determine if the given document should be encrypted</p>	<p>IntelliGard E would access the MAZTable to determine if a document should be encrypted utilizing a PC DOCS API function called “DOCSSQL” (this was also referred to as SQLView). IntelliGard E used the document name (i.e., DOCNUMBER) to check the SECURE flag for the document. The software flow</p>

	<p>for this is found in Exhibit Z-27, pg. 3 and sample source code for the functions to retrieve data from the MAZTable utilizing the DOCS 16 bit API is found in Exhibit Z-23. The first call made to the DOCS API is found in the section “DOCSAPI” on pg. 3. “DOCSBEOpenDocument” determined the fileID of the document currently open in the PC DOCS EDMS (at the time of the transfer of control to IntelliGard E). Using the return value of the fileID, a call to the function “DOCSBeSelectDocument” would set the database record pointer to the proper row in the database. The DOCNUMBER column created in the MAZTable was a “candidate” referential key (see the diagram and column description in Exhibit Z-21, pg. 14). The referential integrity of the database associated the DOCNUMBER to the fileID field that was part of the record where the document name is found. In section “DOCSAPI” pg. 5, the function “DOCSSQLGetColumnValue” was called to get the value for the database column “SECURE”. The return of the function was a 1 if the document record indicated that the document was to be encrypted. If the SECURE field was a binary 1 (yes), then the encryption process event continued. If it was false, the process would save the file in clear text, close the document, and exit the VBA code thus returning control to the PC DOCS EDMS (see Exhibit Z-27, pg. 1 flowchart items (k) – (m). <i>See also</i> Halpern Dec. ¶ 14c; Mahne Dec. ¶ 39c.</p>
<p>if the given document should be encrypted, then</p> <p>retrieve the key name associated with the name of the given document from the first table</p>	<p>If the document should be encrypted, IntelliGard E automatically obtained an encryption key value for the document, as explained below.</p> <p>As explained above, the MAZTable associated documents names (DOCNUMBER) with key names</p>

	<p>(KEYID). Using PC DOCS SQLview, IntelliGard E used the DOCNUMBER to obtain the key name (i.e., KEYID) from the MAZTable (the software process for this is found in Exhibit Z-27, pg. 4).</p> <p>To retrieve the key identity (KEYID) the “DOCSSQLGetColumnValue” (as described above) would again be called with the column name set to “KEYID” which would return the key name associated with the document. <i>See</i> Halpern Dec. ¶ 14c-d; Mahne Dec. ¶ 39c-d.</p>
<p>retrieve the encryption key value associated with the key name from a second table, the second table having at least one encryption key value and at least one key name respectively associated with a one of the encryption key values</p>	<p>As explained above, IntelliGard E worked with smart cards, and the smart cards stored key values in a table, in association with respective key names. The smart card’s key table corresponded to the claimed “second table”. After IntelliGard E obtained the key name from the MAZTable, IntelliGard E would use the KEYID to retrieve the key value from the smart card’s key table. <i>See also</i> Halpern Dec. ¶ 14d; Mahne Dec. ¶ 39d. Sample source code for the reading and administration of the smart card and the smart card’s key table is found in Exhibits Z-11, Z-13, Z-14. In Exhibit Z-11, section “modCardInterface” there are a number of functions that were used to manage the smart card. Public Function “GetUserName” and “GetPassword” on pg. 4 would return the smart card’s security information that protected access to the key table. Also on pg. 4 was the Public Function “GetFileStruct” and on pg. 5 “GetCardType”, “GetAllowedKeyCount” and “GetKeyCount” provided information as to the key table type and size that is found on the inserted smart card. In Exhibit Z-11, section “modMain” the Public Function on pg.1 “WriteKeys” is a sample of how the key names, key values, and the date created were written to the smart card file.</p>

	<p>Exhibit Z-29 is sample source code for the key server that allowed IntelliGard programs to read from the smart card. This is the clsKeyServer object referred to in Exhibit Z-2, section “Igardmain”, pg. 1. To retrieve the key value by keyID (name) from a smart card, a call was made to a Property Get “Get KeyValue” as described in Exhibit Z-29, section “clsKeyServer”, pg. 1. <i>See also</i> Halpern Dec. ¶ 14d; Mahne Dec. ¶ 39d.</p>
<p>cause the given document to be encrypted.</p>	<p>In August 1997 IntelliGard E used the DES algorithm. <i>See</i> Exhibit Z-21, pg. 2 (“All files are encrypted using the Data Encryption Standard (DES) algorithm and an encryption key”). <i>See also</i> Mahne Dec. ¶ 27.</p> <p>After retrieving the key value from the smart card, IntelliGard E automatically caused the document to be encrypted using the encryption key value. <i>See</i> Halpern Dec. ¶ 14d; Mahne Dec. ¶ 39d. Exhibit Z-27, pg. 5 is a flow chart of the encryption process which shows this step. To encrypt a file a call was made to the encryption engine function in the enceng.dll. (This .dll was installed with the IntelliGard E software on the user’s PC.) The EncFile function made a call to “EncryptFile”. The EncryptFile function arguments included the arguments “KeyID” (keyname) and the “CardKey” (key value) that were passed in at the time of call. EncryptFile encrypted the document using an encryption engine and used the DES algorithm. After the document was encrypted, a clear text “header” was added to the beginning of the file (see Exhibit 29, pg. 5, bottom) to indicate the file was encrypted by MAZ IntelliGard and the name of the key used in the encryption. An example of this is in Exhibit Z-19, pg. 5 in the diagram at the bottom of the page. In this file “MAZ\$TECH” indicated it was a MAZ encrypted file, and “A:Classified” was the</p>

	key name used to encrypt the file. <i>See</i> Halpern Dec. ¶ 14d; Mahne Dec. ¶ 39d.
--	---

14. The following claim chart shows how IntelliGard E had all of the features in dependent claims 13, 19, 31, 32, 34, 35, 37-38, 41, 42, 44 and 45:

Claim Language	Corroboration
Claim 13: The process of decrypting documents of claim 29 further comprising providing an electronic document management system comprising a SQL database, a SQL database server and a SQL database client, wherein the electronic document management system performs the detecting step.	<p>IntelliGard E was used for encrypting electronic documents. “IntelliGard is a Windows based product that provides file level security for PC users.” Exhibit Z-21, pg. 2. IntelliGard E integrated with PC DOCS, an off-the-shelf electronic document management system. “The IntelliGard system integration with DOCS Open 3.7.x requires a table in the SQL database(s). The DOCS Open Designer Utility is used to create the new database table in SQL.” Exhibit Z-21, pg. 12 (at top). The PC DOCS EDMS included a SQL database, server, and client, all residing on the IBM compatible PC.</p> <p>In the version of IntelliGard E which we installed, demonstrated and offered to sell to Lyon & Lyon by August 1997, we embedded macros in a MS Word document template. The macros created a new set of file “open”, “save” and “save as” command “buttons” in the users application. Clicking on these command keys would call a function written in VBA (Visual Basic for Applications) that would translate the command into programming events. <i>See</i> Exhibit Z-27, pg. 1 flowchart items (a) and (a1), and Mahne Dec. ¶ 5. This is also described in Exhibit Z-19, pp. 7-8.</p>
Claim 19: The process of decrypting documents of claim 29 further comprising providing a database, the database including an indicator of whether the documents should be decrypted if the indicator in the database does not indicate	<p>IntelliGard E would access the MAZTable to determine if a document should be decrypted utilizing a PC DOCS API function called “DOCSSQL” (this was also referred to as SQLView). IntelliGard E used the document name (i.e., DOCNUMBER) to check the SECURE</p>

<p>that the given document is to be decrypted, determining that the document should not be decrypted.</p>	<p>flag for the document. The software flow for this is found in Exhibit Z-27, pg. 3 and sample source code for the functions to retrieve data from the MAZTable utilizing the DOCS 16 bit API is found in Exhibit Z-23. The first call made to the DOCS API is found in the section “DOCSAPI” on pg. 3. “DOCSBEOpenDocument” determined the fileID of the document currently open in the PC DOCS EDMS (at the time of the transfer of control to IntelliGard E). Using the return value of the fileID, a call to the function “DOCSBeSelectDocument” would set the database record pointer to the proper row in the database. The DOCNUMBER column created in the MAZTable was a “candidate” referential key (see the diagram and column description in Exhibit Z-21, pg. 14). The referential integrity of the database associated the DOCNUMBER to the fileID field that was part of the record where the document name is found. In section “DOCSAPI” pg. 5, the function “DOCSSQLGetColumnValue” was called to get the value for the database column “SECURE”. The return of the function was a 1 if the document record indicated that the document was encrypted.</p> <p>If the SECURE field was a binary 1 (yes), then the decryption process event continued. If it was false, the process would determine that the document should not be decrypted, and exit the VBA code thus returning control to the PC DOCS EDMS (see Exhibit Z-27, pg. 1 flowchart items (k) – (m). <i>See also</i> Halpern Dec. ¶ 14c; Mahne Dec. ¶ 39c).</p>
<p>Claim 31: The process of decrypting documents of claim 29 further comprising decrypting the given document with a DES algorithm.</p>	<p>Intelligard E utilized the DES algorithm for encrypting and decrypting documents. (See Exhibit Z-21, page 2).</p>
<p>Claim 32: The process of decrypting documents of claim 29 wherein the second table is stored in a smart card.</p>	<p>As explained above, IntelliGard E worked with smart cards, and the smart cards stored key values in a table, in association</p>

	with respective key names. The smart card's key table corresponded to the claimed "second table". <i>See also</i> Halpern Dec. ¶ 14d; Mahne Dec. ¶ 39d. Sample source code for the reading and administration of the smart card and the smart card's key table is found in Exhibits Z-11, Z-13, Z-14.
Claim 35: The computer program product of claim 33, the program code further for causing the processor to decrypt the given document with a DES_algorithm	Intelligard E utilized the DES algorithm for encrypting and decrypting documents. (See Exhibit Z-21, page 2).
Claim 35: A general purpose computer system comprising the computer program product of claim 33.	IntelliGard E was used for decrypting electronic documents. "IntelliGard is a Windows based product that provides file level security for PC users." Exhibit Z-21, pg. 2. IntelliGard E integrated with PC DOCS, an off-the-shelf electronic document management system. "The IntelliGard system integration with DOCS Open 3.7.x requires a table in the SQL database(s). The DOCS Open Designer Utility is used to create the new database table in SQL." Exhibit Z-21, pg. 12 (at top). IntelliGard E required a PC running Windows and PC DOCS. <i>See</i> Exhibit Z-21, pg. 3 ("System Requirements . . . IBM compatible 486 PC . . . Microsoft Windows® 95, 98, NT4.0"). The Windows PC was a general purpose computer, and it is well known that application programs such as MS Word ran on Windows PCs and processed electronic documents. These processes included opening, closing, and saving the electronic documents.
Claim 37: The computer program product of claim 33, the program code further for causing the processor to obtain decryption key values from a portable data storage device. Claim 38: The computer program product of claim 33 wherein the second table is stored in a smart card.	As explained above, IntelliGard E worked with smart cards, and the smart cards stored key values in a table, in association with respective key names. The smart card's key table corresponded to the claimed "second table". After IntelliGard E obtained the key name from the MAZTable, IntelliGard E would use the KEYID to retrieve the key value from the

	<p>smart card's key table. <i>See also</i> Halpern Dec. ¶ 14d; Mahne Dec. ¶ 39d. Sample source code for the reading and administration of the smart card and the smart card's key table is found in Exhibits Z-11, Z-13, Z-14. In Exhibit Z-11, section "modCardInterface" there are a number of functions that were used to manage the smart card. Public Function "GetUserName" and "GetPassword" on pg. 4 would return the smart card's security information that protected access to the key table. Also on pg. 4 was the Public Function "GetFileStruct" and on pg. 5 "GetCardType", "GetAllowedKeyCount" and "GetKeyCount" provided information as to the key table type and size that is found on the inserted smart card. In Exhibit Z-11, section "modMain" the Public Function on pg.1 "WriteKeys" is a sample of how the key names, key values, and the date created were written to the smart card file.</p> <p>Exhibit Z-29 is sample source code for the key server that allowed IntelliGard programs to read from the smart card. This is the clsKeyServer object referred to in Exhibit Z-2, section "Igarmain", pg. 1. To retrieve the key value by keyID (name) from a smart card, a call was made to a Property Get "Get KeyValue" as described in Exhibit Z-29, section "clsKeyServer", pg. 1. <i>See also</i> Halpern Dec. ¶ 14d; Mahne Dec. ¶ 39d.</p>
Claim 41: The computer program product of claim 40, the program code further for causing the processor to encrypt the given document with a DES algorithm.	Intelligard E utilized the DES algorithm for encrypting and decrypting documents. (See Exhibit Z-21, page 2).
Claim 42: A general purpose computer system comprising the computer program product of claim 40.	IntelliGard E was used for decrypting electronic documents. "IntelliGard is a Windows based product that provides file level security for PC users." Exhibit Z-21, pg. 2. IntelliGard E integrated with PC DOCS, an off-the-shelf electronic document management system. "The


	<p>IntelliGard system integration with DOCS Open 3.7.x requires a table in the SQL database(s). The DOCS Open Designer Utility is used to create the new database table in SQL.” Exhibit Z-21, pg. 12 (at top). IntelliGard E required a PC running Windows and PC DOCS. <i>See</i> Exhibit Z-21, pg. 3 (“System Requirements . . . IBM compatible 486 PC . . . Microsoft Windows® 95, 98, NT4.0”). The Windows PC was a general purpose computer, and it is well known that application programs such as MS Word ran on Windows PCs and processed electronic documents. These processes included opening, closing, and saving the electronic documents.</p>
<p>Claim 44: The computer program product of claim 40, the program code further for causing the processor to obtain encryption key values from a portable data storage device.</p> <p>Claim 45: The computer program product of claim 40 wherein the second table is stored in a smart card.</p>	<p>As explained above, IntelliGard E worked with smart cards, and the smart cards stored key values in a table, in association with respective key names. The smart card’s key table corresponded to the claimed “second table”. After IntelliGard E obtained the key name from the MAZTable, IntelliGard E would use the KEYID to retrieve the key value from the smart card’s key table. <i>See also</i> Halpern Dec. ¶ 14d; Mahne Dec. ¶ 39d. Sample source code for the reading and administration of the smart card and the smart card’s key table is found in Exhibits Z-11, Z-13, Z-14. In Exhibit Z-11, section “modCardInterface” there are a number of functions that were used to manage the smart card. Public Function “GetUserName” and “GetPassword” on pg. 4 would return the smart card’s security information that protected access to the key table. Also on pg. 4 was the Public Function “GetFileStruct” and on pg. 5 “GetCardType”, “GetAllowedKeyCount” and “GetKeyCount” provided information as to the key table type and size that is found on the inserted smart card. In Exhibit Z-11, section “modMain” the Public Function on pg.1 “WriteKeys” is a</p>

	<p>sample of how the key names, key values, and the date created were written to the smart card file.</p> <p>Exhibit Z-29 is sample source code for the key server that allowed IntelliGard programs to read from the smart card. This is the clsKeyServer object referred to in Exhibit Z-2, section "Igardenmain", pg. 1. To retrieve the key value by keyID (name) from a smart card, a call was made to a Property Get "Get KeyValue" as described in Exhibit Z-29, section "clsKeyServer", pg. 1. <i>See also</i> Halpern Dec. ¶ 14d; Mahne Dec. ¶ 39d.</p>
--	--

Affirmation

15. All exhibits attached to this declaration are true and correct copies. Each exhibit either speaks for itself, or I have described it herein.
16. I further declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of any patent issuing from the '246 application.

Date: 4/10/2006




Stephen Zizzi

EXHIBIT Z-2

IntelliGard Main Console

Program Version 2.3

Declaration of Stephen Zizzi



IntelliGard 2.3



Copyright 1996-98 Maz Technologies, Inc.



IntelliGard Smart Suite
System Console

Version 1.5

Copyright 1996-97, Maz Technologies, Inc.

WARNING! For use in the U.S. and Canada only. Export
license required. For more information contact Maz
Technologies or the U.S. Commerce Department Bureau
of Export Controls, Washington, D.C.



[Redacted]

[Redacted]

[Redacted]

minutes



[Redacted]

[Redacted]

[Redacted]

Current User Name:

[REDACTED]

Available Keys:

[REDACTED]

[REDACTED]

User Network Login Name:



User Computer Name:

Available Keys:



User Mode

2:53 PM

9/26/2004

WARNING - a file that is shredded cannot be recovered!



File to be shred:



Igardmain - 1

' IntelliGard E Series - S.J. Zizzi, December 1996.
' IntelliGard Smart Suite System Console

Option Explicit

(Initialize objects

Public fMainForm As frmIgardMain

Public oServer As New clsKeyServer
Public oAccess As New clsAccessServer

' Initialize program flags and global variables
Public hwnd&

Public SOpenFlag As Boolean
Public SReadPassword As String
Public UserReadPassword As String
Public CurrentUserName As String
Public ValidUserLogin As Boolean
Public SystemStarted As Boolean
Public SSessionOpen As Boolean
Public IsSystemAdmin As Boolean
Public LogFileType As Integer
Public UserType As String
Public UserCardID As String

' Initialize File Name Strings

Public zzWindowsDir As String * 128
Public zzSystemDir As String * 128
'Public sWorkingPath As String
Public CfgFile As String * 128
Public UserLog As String * 128
Public CryptoLog As String * 128
Public ServerLog As String * 128

Public zzKeySelected As String
Public InFileName As String
Public OutFileName As String * 128

Dim ParseFileName As String

Dim ParseFileExt As String
Dim ParseFilePath As String
Dim ParseFilePathEnd As String

' Initialize zzSys Environment Vars

Public zzCompress As Integer
Public zzConfirmRO As Integer
Public zzShred As Integer
Public zzDefaultPath As String
Public zzLongNameOpts As Boolean
Public zzLongFileName As Boolean
Public zzLongFileExt As Boolean
Public zzShortNameOpts As Boolean
Public zzShortFileName As Boolean
Public zzShortFileExt As Boolean

Public zzFileMode As Long
Public cmode As String

Public zzCustomExt As Boolean
Public zzNoNameChange As Boolean
Public zzNetLogIn As Long
Public zzScreenSaver As Boolean

```
Public zzNoStandby As Boolean
Public zzExitSystem As Boolean
Public zzShutDown As Boolean
Public zzStartSaver As Boolean
Public zzIgnorSC As Boolean
```

```
Public zzAuthenticate As Integer
Public zzLaunchApp As Integer
Public zzTxtTimeOut As Integer
Public zzTxtServerName As String
Public zzTxtCustomExt As String * 3
Public zzTxtScreenSaverName As String
```

```
Public sKeyValue As String
```

```
Declare Function IsIconic Lib "user32" (ByVal hwnd As Long) As Long
```

```
Declare Function GetWindowsDirectory Lib "kernel32" Alias "GetWindowsDirectoryA" (ByVal lpBuffer As String, ByVal nSize As Long) As Long
Declare Function GetSystemDirectory Lib "kernel32" Alias "GetSystemDirectoryA" (ByVal lpBuffer As String, ByVal nSize As Long) As Long
Declare Function FindWindow Lib "user32" Alias "FindWindowA" (ByVal lpClassName As String, ByVal lpWindowName As String) As Long
Declare Function SetForegroundWindow Lib "user32" (ByVal hwnd As Long) As Long
Declare Function SetActiveWindow Lib "user32" (ByVal hwnd As Long) As Long
Declare Function SetFocus Lib "user32" (ByVal hwnd As Long) As Long
Declare Function OpenIcon Lib "user32" (ByVal hwnd As Long) As Long
Declare Function GetActiveWindow Lib "user32" () As Long
```

```
Declare Function ExitWindows Lib "user32" (ByVal dwReserved As Long, ByVal uReturnCode As Long) As Long
Declare Function ExitWindowsEx Lib "user32" (ByVal uFlags As Long, ByVal dwReserved As Long) As Long
```

```
Public Declare Function ShowWindow Lib "user32" (ByVal hwnd As Long, ByVal nCmdShow As Long) As Long
Public Declare Function CloseWindow Lib "user32" (ByVal hwnd As Long) As Long
Public Declare Function GetFileAttributes Lib "kernel32" Alias "GetFileAttributesA" (ByVal lpFileName As String) As Long
Public Declare Function SetFileAttributes Lib "kernel32" Alias "SetFileAttributesA" (ByVal lpFileName As String, ByVal dwFileAttributes As Long) As Long
Public Declare Function GetPrivateProfileString Lib "kernel32" Alias "GetPrivateProfileStringA" (ByVal lpApplicationName As String, ByVal lpKeyName As String, ByVal lpDefault As String, ByVal lpReturnedString As String, ByVal nSize As Long, ByVal lpFileName As String) As Long
Public Declare Function WritePrivateProfileString Lib "kernel32" Alias "WritePrivateProfileStringA" (ByVal lpApplicationName As String, ByVal lpKeyName As Any, ByVal lpString As Any, ByVal lpFileName As String) As Long
Public Declare Function GetUserName Lib "advapi32.dll" Alias "GetUserNameA" (ByVal lpBuffer As String, nSize As Long) As Long
' Constants
```

```
Public Const FILE_ATTRIBUTE_ARCHIVE = &H20
Global Const FILE_ATTRIBUTE_COMPRESSED = &H800
Public Const FILE_ATTRIBUTE_DIRECTORY = &H10
Public Const FILE_ATTRIBUTE_HIDDEN = &H2
Public Const FILE_ATTRIBUTE_NORMAL = &H80
Global Const FILE_ATTRIBUTE_READONLY = &H1
Global Const FILE_ATTRIBUTE_SYSTEM = &H4
Public Const FILE_ATTRIBUTE_TEMPORARY = &H100
Public Const SW_SHOWNORMAL = 1
```

```
Global Const vbRightButton = 2
Public YNStyle As String
```

```
' Declare Encryption Engine Functions
```

```
Public Declare Function Shred Lib "enceng20.dll" (ByVal filename As String) As Long
Public Declare Function EncFile Lib "enceng20.dll" (ByVal filename As String, ByVal postfilename As String, ByVal key_ID As String, ByVal card_key As String) As Long
Public Declare Function EncFileNoDestroy Lib "enceng20.dll" (ByVal filename As String, ByVal postf
```


Igardmain - 3

```
ilename As String, ByVal key_ID As String, ByVal card_key As String) As Long
Public Declare Function PrepDec Lib "enceng20.dll" (ByVal filename As String, ByVal key_ID As String) As Long
Public Declare Function DecFileNoDestroy Lib "enceng20.dll" (ByVal postfilename As String, ByVal key_ID As String) As Long
Public Declare Function DecFile Lib "enceng20.dll" (ByVal postfilename As String, ByVal Key_value As String) As Long
```

Sub Main()

```
Set fMainForm = New frmIgardMain
Dim fsplash As New frmsplash
Dim X As Integer
Dim nError As Long
```

```
'***** Check if app is already running
Call IsRunning
```

```
'***** Get configuration from ini file
Call GetDirs
Call GetInitCfg
```

```
'***** Check for valid login
```

```
' If oAccess.ValidateUser("z1001941") Then
    Load fsplash
    fsplash.Show
    fsplash.Refresh
' Else
'     nError = MsgBox("Smart Card Server Refused Access, Please Contact System Administrator", vbCritical + vbOKOnly, "IntelliGard Smart Suite")
' End
' End If
fsplash.Refresh

Call UpdateUserType
Unload fsplash
fMainForm.Show vbModal
' UserCardID = oAccess.sCardUserID
```

End Sub

Public Sub GetInitCfg()

```
Dim zApp As String
Dim zFile As String * 255
Dim zstring As String
Dim zKeyName As String
Dim zDefault As String
Dim zReturn As String * 32
Dim zSize As Long
Dim RetOK As Long
```

```
zApp = "General"
zFile = CfgFile
```

```
zResult = "1"
zSize = 32
zstring = "6"
```

```
zKeyName = "zzCompress"
RetOK = GetPrivateProfileString(zApp, zKeyName, zDefault, zReturn, zSize, zFile)
```

```

zzCompress = zReturn

zKeyName = "zzConfirmRO"
RetOK = GetPrivateProfileString(zApp, zKeyName, zDefault, zReturn, zSize, zFile)
zzConfirmRO = zReturn

zKeyName = "zzShred"
RetOK = GetPrivateProfileString(zApp, zKeyName, zDefault, zReturn, zSize, zFile)
zzShred = zReturn

zKeyName = "zzDefaultPath"
RetOK = GetPrivateProfileString(zApp, zKeyName, zDefault, zReturn, zSize, zFile)
zzDefaultPath = zReturn

zKeyName = "zzLongFileName"
RetOK = GetPrivateProfileString(zApp, zKeyName, zDefault, zReturn, zSize, zFile)
zzLongFileName = zReturn

zKeyName = "zzLongFileExt"
RetOK = GetPrivateProfileString(zApp, zKeyName, zDefault, zReturn, zSize, zFile)
zzLongFileExt = zReturn

zKeyName = "zzShortFileName"
RetOK = GetPrivateProfileString(zApp, zKeyName, zDefault, zReturn, zSize, zFile)
zzShortFileName = zReturn

zKeyName = "zzShortFileExt"
RetOK = GetPrivateProfileString(zApp, zKeyName, zDefault, zReturn, zSize, zFile)
zzShortFileExt = zReturn

zKeyName = "zzNoNameChange"
RetOK = GetPrivateProfileString(zApp, zKeyName, zDefault, zReturn, zSize, zFile)
zzNoNameChange = zReturn

zKeyName = "zzFileMode"
RetOK = GetPrivateProfileString(zApp, zKeyName, zDefault, zReturn, zSize, zFile)
zzFileMode = Val(zReturn)

zKeyName = "zzCustomExt"
RetOK = GetPrivateProfileString(zApp, zKeyName, zDefault, zReturn, zSize, zFile)
zzCustomExt = zReturn

zKeyName = "zzTxtCustomExt"
RetOK = GetPrivateProfileString(zApp, zKeyName, zDefault, zReturn, zSize, zFile)
zzTxtCustomExt = zReturn

zKeyName = "zzNetLogIn"
RetOK = GetPrivateProfileString(zApp, zKeyName, zDefault, zReturn, zSize, zFile)
zzNetLogIn = zReturn

zKeyName = "zzTxtScreenSaverName"
zReturn = Space(32)
RetOK = GetPrivateProfileString(zApp, zKeyName, zDefault, zReturn, zSize, zFile)
zzTxtScreenSaverName = zReturn

zKeyName = "zzScreenSaver"
zReturn = Space(7)
RetOK = GetPrivateProfileString(zApp, zKeyName, zDefault, zReturn, zSize, zFile)
zzScreenSaver = zReturn

zKeyName = "zzNoStandby"
RetOK = GetPrivateProfileString(zApp, zKeyName, zDefault, zReturn, zSize, zFile)
zzNoStandby = zReturn

zKeyName = "zzExitSystem"
RetOK = GetPrivateProfileString(zApp, zKeyName, zDefault, zReturn, zSize, zFile)
zzExitSystem = zReturn

zKeyName = "zzShutDown"
RetOK = GetPrivateProfileString(zApp, zKeyName, zDefault, zReturn, zSize, zFile)

```

```
zzShutDown = zReturn
```

```
zKeyName = "zzStartSaver"
```

```
RetOK = GetPrivateProfileString(zApp, zKeyName, zDefault, zReturn, zSize, zFile)
```

```
zzStartSaver = zReturn
```

```
zKeyName = "zzIgnorSC"
```

```
RetOK = GetPrivateProfileString(zApp, zKeyName, zDefault, zReturn, zSize, zFile)
```

```
zzIgnorSC = zReturn
```

```
zKeyName = "cmode"
```

```
RetOK = GetPrivateProfileString(zApp, zKeyName, zDefault, zReturn, zSize, zFile)
```

```
cmode = Val(zReturn)
```

```
zKeyName = "zzAuthenticate"
```

```
RetOK = GetPrivateProfileString(zApp, zKeyName, zDefault, zReturn, zSize, zFile)
```

```
zzAuthenticate = zReturn
```

```
zKeyName = "zzLaunchApp"
```

```
RetOK = GetPrivateProfileString(zApp, zKeyName, zDefault, zReturn, zSize, zFile)
```

```
zzLaunchApp = zReturn
```

```
zKeyName = "zzTxtTimeOut"
```

```
zReturn = Space(3)
```

```
RetOK = GetPrivateProfileString(zApp, zKeyName, zDefault, zReturn, zSize, zFile)
```

```
zzTxtTimeOut = zReturn
```

```
zKeyName = "zzTxtServerName"
```

```
zReturn = Space(32)
```

```
RetOK = GetPrivateProfileString(zApp, zKeyName, zDefault, zReturn, zSize, zFile)
```

```
zzTxtServerName = zReturn
```

```
End Sub
```

```
Public Function sTrimNulls(sText As String) As String
```

```
Dim nPos As Integer
```

```
Dim nNull As Integer
```

```
Dim sNewText As String
```

```
nNull = InStr(sText, Chr$(0))
```

```
If nNull = 0 Then
```

```
    sTrimNulls = sText
```

```
Else
```

```
    nPos = 1
```

```
    While Asc(Mid(sText, nPos, 1)) <> 0
```

```
        sNewText = sNewText & Mid(sText, nPos, 1)
```

```
        nPos = nPos + 1
```

```
    Wend
```

```
    sTrimNulls = sNewText
```

```
End If
```

```
End Function
```

```
Function sTrimNulls1(sText As String) As String
```

```
Dim n As Integer
```

```
Dim bFound As Boolean
```

```
Dim sNew As String
```

```
n = 1
```

```
If Asc(Mid(sText, n, 1)) = 0 Then
```

```
    bFound = True
```

```
Else
```

```
    n = n + 1
```

```
End If
```

```
Loop Until bFound Or n > Len(sText)
```

```

If bFound Then
    sTrimNulls1 = Left$(sText, n - 1)
Else
    sTrimNulls1 = sText
End If

```

End Function

Public Sub UpdateUserType()

UserType = UCase(oAccess.UserType)

Select Case UserType

Case "F"

```

fMainForm.StatusBar1.Panels.Item(1).Text = "SA Mode"
fMainForm.mnuSAAdmin.Visible = True
fMainForm.mnuCardman.Visible = True

```

Case "S"

```

fMainForm.StatusBar1.Panels.Item(1).Text = "SA Mode"
fMainForm.mnuSAAdmin.Visible = True
fMainForm.mnuCardman.Visible = False

```

Case "U"

```

fMainForm.StatusBar1.Panels.Item(1).Text = "User Mode"
fMainForm.mnuSAAdmin.Visible = False
fMainForm.mnuCardman.Visible = False

```

Case Else

```

fMainForm.StatusBar1.Panels.Item(1).Text = "User Mode"
fMainForm.mnuSAAdmin.Visible = False
fMainForm.mnuCardman.Visible = False

```

End Select

E Sub

Public Sub EncNamePrep()

Select Case zzFileMode

Case 1

```

Call ParseName
OutFileName = ParseFileName & "-maz" & ParseFileExt

```

Case 2

```

OutFileName = InFileName & ".maz"

```

Case 3

```

Call ParseName
OutFileName = ParseFilePath & "\!" & ParseFilePathEnd

```

Case 5

```

Call ParseName
OutFileName = ParseFileName & "." & zzTxtCustomExt

```

Case 4

```

Call ParseName
OutFileName = ParseFileName & ".maz"

```

Case 6

```

OutFileName = InFileName

```

Case Else

```

OutFileName = InFileName

```

End Select

E Sub

Public Sub ParseName()

```

Dim ParseChar As String
Dim X As Integer
Dim Y As Integer
Dim FoundIt As Boolean

```

Igardmain - 7

```
Dim ExtChar As String
```

```
X = 1  
FoundIt = False
```

```
Do Until FoundIt
```

```
    ParseChar = Mid(InFileName, X, 1)
```

```
    If ParseChar = "\" Then  
        ParseFilePath = Left(InFileName, X - 1)  
        Y = X
```

```
    End If
```

```
    If ParseChar = "." Then  
        ParseFileName = Left(InFileName, X - 1)  
        ParseFileExt = Right(InFileName, (Len(InFileName) - (X - 1)))  
        ParseFilePathEnd = Right(InFileName, (Len(InFileName) - Y))  
        FoundIt = True
```

```
        Exit Sub  
    End If
```

```
    X = X + 1
```

```
Loop
```

```
End Sub
```

```
Public Sub GetDirs()
```

```
Dim WinReturn As Long  
Dim SysReturn As Long  
Dim wsize As Long  
Dim ssize As Long  
Dim zReturn As String * 255  
Dim ReturnSz As Long  
Dim zApp As String  
Dim zDefault As String  
Dim zSize As Long  
Dim zKeyName As String  
Dim sWorkingPath As String
```

```
wsize = 250  
ssize = 250
```

```
WinReturn = GetWindowsDirectory(zzWindowsDir, wsize)  
SysReturn = GetSystemDirectory(zzSystemDir, ssize)  
CfgFile = Left(zzWindowsDir, WinReturn) & "\mfc11.dat"  
zFile = Left(zzWindowsDir, WinReturn) & "\msysstate.ini"
```

```
sWorkingPath = App.Path  
App.HelpFile = App.Path & "\intelligard help.hlp"
```

```
UserLog = Left(zzWindowsDir, WinReturn) & "\mazusr.log"  
CryptoLog = Left(zzWindowsDir, WinReturn) & "\mazfile.log"  
ServerLog = Left(zzWindowsDir, WinReturn) & "\mazsrv.log"
```

```
End Sub
```

```
Private Sub IsRunning()
```

```
Dim Appname$  
Dim hwndold$, hwndopen$  
Appname = "IntelliGard Console"  
hwnd = FindWindow(vbNullString, Appname)
```

```
If hwnd <> 0 Then  
    hwndold = SetForegroundWindow(hwnd)  
    hwndopen = OpenIcon(hwnd)  
End
```

Igardmain - 8

End If

End Sub

(

frmAbout - 1

```
' Reg Key Security Options...
Const KEY_ALL_ACCESS = &H2003F
```

```
' Reg Key ROOT Types...
Const HKEY_LOCAL_MACHINE = &H80000002
Const ERROR_SUCCESS = 0
Const REG_SZ = 1 ' Unicode nul terminated string
Const REG_DWORD = 4 ' 32-bit number
```

```
Const gREGKEYSYSINFOLOC = "SOFTWARE\Microsoft\Shared Tools Location"
Const gREGVALSYSINFOLOC = "MSINFO"
Const gREGKEYSYSINFO = "SOFTWARE\Microsoft\Shared Tools\MSINFO"
Const gREGVALSYSINFO = "PATH"
```

```
Private Declare Function RegOpenKeyEx Lib "advapi32" Alias "RegOpenKeyExA" (ByVal hKey As Long, ByVal lpSubKey As String, ByVal ulOptions As Long, ByVal samDesired As Long, ByRef phkResult As Long) As Long
Private Declare Function RegQueryValueEx Lib "advapi32" Alias "RegQueryValueExA" (ByVal hKey As Long, ByVal lpValueName As String, ByVal lpReserved As Long, ByRef lpType As Long, ByVal lpData As String, ByRef lpcbData As Long) As Long
Private Declare Function RegCloseKey Lib "advapi32" (ByVal hKey As Long) As Long
```

```
Private Sub Form_Load()
    lblVersion.Caption = "Version " & App.Major & "." & App.Minor & "." & App.Revision
    lblTitle.Caption = App.Title
    If UserType = "S" Or UserType = "F" Then
        cmdSysInfo.Visible = True
    Else
        cmdSysInfo.Visible = False
    End If
End Sub
```

E Sub

```
Private Sub cmdSysInfo_Click()
    Call StartSysInfo
End Sub
```

```
Private Sub cmdOK_Click()
    Unload Me
End Sub
```

```
Public Sub StartSysInfo()
    On Error GoTo SysInfoErr
```

```
    Dim rc As Long
    Dim SysInfoPath As String
```

```
    ' Try To Get System Info Program Path\Name From Registry...
    If GetKeyValue(HKEY_LOCAL_MACHINE, gREGKEYSYSINFO, gREGVALSYSINFO, SysInfoPath) Then
        ' Try To Get System Info Program Path Only From Registry...
    ElseIf GetKeyValue(HKEY_LOCAL_MACHINE, gREGKEYSYSINFOLOC, gREGVALSYSINFOLOC, SysInfoPath)
```

```
Then
    ' Validate Existence Of Known 32 Bit File Version
    If (Dir(SysInfoPath & "\MSINFO32.EXE") <> "") Then
        SysInfoPath = SysInfoPath & "\MSINFO32.EXE"
```

```
    ' Error - File Can Not Be Found...
    Else
```

GoTo SysInfoErr

End If

' Error - Registry Entry Can Not Be Found...

Else

GoTo SysInfoErr

End If

Call Shell(SysInfoPath, vbNormalFocus)

Exit Sub

SysInfoErr:

MsgBox "System Information Is Unavailable At This Time", vbOKOnly

End Sub

Public Function GetKeyValue(KeyRoot As Long, keyname As String, SubKeyRef As String, ByRef KeyVal As String) As Boolean

Dim i As Long ' Loop Counter

Dim rc As Long ' Return Code

Dim hKey As Long ' Handle To An Open Registry Key

Dim hDepth As Long

Dim KeyValType As Long ' Data Type Of A Registry Key

Dim tmpVal As String ' Tempory Storage For A Registry K

ey Value

Dim KeyValSize As Long ' Size Of Registry Key Variable

 ' Open RegKey Under KeyRoot {HKEY_LOCAL_MACHINE...}

rc = RegOpenKeyEx(KeyRoot, keyname, 0, KEY_ALL_ACCESS, hKey) ' Open Registry Key

If (rc <> ERROR_SUCCESS) Then GoTo GetKeyError ' Handle Error...

tmpVal = String\$(1024, 0) ' Allocate Variable Space

KeyValSize = 1024 ' Mark Variable Size

 ' Retrieve Registry Key Value...

rc = RegQueryValueEx(hKey, SubKeyRef, 0, KeyValType, tmpVal, KeyValSize) ' Get/Create K

ey Value

If (rc <> ERROR_SUCCESS) Then GoTo GetKeyError ' Handle Errors

If (Asc(Mid(tmpVal, KeyValSize, 1)) = 0) Then ' Win95 Adds Null Terminated Strin

g...

tmpVal = Left(tmpVal, KeyValSize - 1) ' Null Found, Extract From Str

ing

Else ' WinNT Does NOT Null Terminate St

ring...

tmpVal = Left(tmpVal, KeyValSize) ' Null Not Found, Extract Stri

ng Only

End If

 ' Determine Key Value Type For Conversion...

Select Case KeyValType ' Search Data Types...

Case REG_SZ ' String Registry Key Data Type

KeyVal = tmpVal ' Copy String Value

Case REG_DWORD ' Double Word Registry Key Data Ty

pe

For i = Len(tmpVal) To 1 Step -1 ' Convert Each Bit

KeyVal = KeyVal + Hex(Asc(Mid(tmpVal, i, 1))) ' Build Value Char. By Cha

r.

frmAbout - 3

```
Next
KeyVal = Format$("&h" + KeyVal)
```

```
' Convert Double Word To Strin
```

9

```
End Select
```

(

```
GetKeyValue = True
rc = RegCloseKey(hKey)
Exit Function
```

```
' Return Success
' Close Registry Key
' Exit
```

```
GetKeyError: ' Cleanup After An Error Has Occured...
```

```
KeyVal = ""
```

```
GetKeyValue = False
```

```
rc = RegCloseKey(hKey)
```

```
End Function
```

```
' Set Return Val To Empty String
' Return Failure
' Close Registry Key
```

frmConfig - 1

Private Sub CfgUpdate()

'Dim SetAttOK As Boolean

If OptT1LongFileName.Value Then
 zzFileMode = 1
End If

If OptT1LongFileExt.Value Then
 zzFileMode = 2
End If

If OptT1ShortFileName.Value Then
 zzFileMode = 3
End If

If OptT1CustomExt.Value Then
 zzFileMode = 5
End If

If OptT1ShortFileExt.Value Then
 zzFileMode = 4
End If

If OptT1NoNameChange.Value Then
 zzFileMode = 6
End If

Dim zApp As String
Dim zstring As String
Dim zKeyName As String
Dim zDefault As String
Dim zReturn As String
Dim zSize As Long
Dim RetOK As Long

zApp = "General"
zDefault = "0"
zSize = 32

zzCompress = ChkT0Compress.Value
zstring = ChkT0Compress.Value
zKeyName = "zzCompress"
RetOK = WritePrivateProfileString(zApp, zKeyName, zstring, CfgFile)

zzLongFileName = OptT1LongFileName.Value
zstring = OptT1LongFileName.Value
zKeyName = "zzLongFileName"
RetOK = WritePrivateProfileString(zApp, zKeyName, zstring, CfgFile)

zzLongFileExt = OptT1LongFileExt.Value
zstring = OptT1LongFileExt.Value
zKeyName = "zzLongFileExt"
RetOK = WritePrivateProfileString(zApp, zKeyName, zstring, CfgFile)

zzShortFileName = OptT1ShortFileName.Value
zstring = OptT1ShortFileName.Value
zKeyName = "zzShortFileName"
RetOK = WritePrivateProfileString(zApp, zKeyName, zstring, CfgFile)

zzShortFileExt = OptT1ShortFileExt.Value
zstring = OptT1ShortFileExt.Value
zKeyName = "zzShortFileExt"
RetOK = WritePrivateProfileString(zApp, zKeyName, zstring, CfgFile)

zzCustomExt = OptT1CustomExt.Value
zstring = OptT1CustomExt.Value
zKeyName = "zzShortFileExt"

frmConfig - 2

```
RetOK = WritePrivateProfileString(zApp, zKeyName, zstring, CfgFile)
```

```
zzNoNameChange = OptT1NoNameChange.Value
```

```
zstring = OptT1NoNameChange.Value
```

```
zKeyName = "zzNoNameChange"
```

```
RetOK = WritePrivateProfileString(zApp, zKeyName, zstring, CfgFile)
```

```
zzNetLogIn = OptT2NetLogIn.Value
```

```
zstring = OptT2NetLogIn.Value
```

```
zKeyName = "zzNetLogIn"
```

```
RetOK = WritePrivateProfileString(zApp, zKeyName, zstring, CfgFile)
```

```
zzScreenSaver = OptT2ScreenSaver.Value
```

```
zstring = OptT2ScreenSaver.Value
```

```
zKeyName = "zzScreenSaver"
```

```
RetOK = WritePrivateProfileString(zApp, zKeyName, zstring, CfgFile)
```

```
zzNoStandby = OptT2NoStandby.Value
```

```
zstring = OptT2NoStandby.Value
```

```
zKeyName = "zzNoStandby"
```

```
RetOK = WritePrivateProfileString(zApp, zKeyName, zstring, CfgFile)
```

```
zzExitSystem = OptT2ExitSystem.Value
```

```
zstring = OptT2ExitSystem.Value
```

```
zKeyName = "zzExitSystem"
```

```
RetOK = WritePrivateProfileString(zApp, zKeyName, zstring, CfgFile)
```

```
zzShutDown = OptT2ShutDown.Value
```

```
zstring = OptT2ShutDown.Value
```

```
zKeyName = "zzShutDown"
```

```
RetOK = WritePrivateProfileString(zApp, zKeyName, zstring, CfgFile)
```

```
zzStartSaver = OptT2StartSaver.Value
```

```
zstring = OptT2StartSaver.Value
```

```
zKeyName = "zzStartSaver"
```

```
RetOK = WritePrivateProfileString(zApp, zKeyName, zstring, CfgFile)
```

```
zzIgnorSC = OptT2IgnorSC.Value
```

```
zstring = OptT2IgnorSC.Value
```

```
zKeyName = "zzIgnorSC"
```

```
RetOK = WritePrivateProfileString(zApp, zKeyName, zstring, CfgFile)
```

```
zzTxtTimeOut = TxtTimeOut.Text
```

```
zstring = TxtTimeOut.Text
```

```
zKeyName = "zzTxtTimeOut"
```

```
RetOK = WritePrivateProfileString(zApp, zKeyName, zstring, CfgFile)
```

```
zzConfirmEncrypt = ChkT0ConfirmEncrypt.Value
```

```
zstring = ChkT0ConfirmEncrypt.Value
```

```
zKeyName = "zzConfirmEncrypt"
```

```
RetOK = WritePrivateProfileString(zApp, zKeyName, zstring, CfgFile)
```

```
zzConfirmRO = ChkT0ConfirmRO.Value
```

```
zstring = ChkT0ConfirmRO.Value
```

```
zKeyName = "zzConfirmRO"
```

```
RetOK = WritePrivateProfileString(zApp, zKeyName, zstring, CfgFile)
```

```
zzShred = ChkT0Shred.Value
```

```
zstring = ChkT0Shred.Value
```

```
zKeyName = "zzShred"
```

```
RetOK = WritePrivateProfileString(zApp, zKeyName, zstring, CfgFile)
```

```
' Write Text Values
```

```
{ zzTxtCustomExt = TxtCustomExt.Text
```

```
zstring = TxtCustomExt.Text
```

```
zKeyName = "zzTxtCustomExt"
```

```
RetOK = WritePrivateProfileString(zApp, zKeyName, zstring, CfgFile)
```

```
zzTxtScreenSaverName = TxtScreenSaver.Text
```

frmConfig - 3

```
zstring = TxtScreenSaver.Text
zKeyName = "zzTxtScreenSaverName"
RetOK = WritePrivateProfileString(zApp, zKeyName, zstring, CfgFile)
```

```
'           Update system modes
zstring = zzFileMode
zKeyName = "zzFileMode"
RetOK = WritePrivateProfileString(zApp, zKeyName, zstring, CfgFile)
```

```
If zzStartSaver Then
    cmode = "A"
End If
```

```
If zzShutDown Then
    cmode = "B"
End If
```

```
If zzIgnorSC Then
    cmode = "C"
End If
```

```
zstring = cmode
zKeyName = "cmode"
zDefault = "A"
RetOK = WritePrivateProfileString(zApp, zKeyName, zstring, CfgFile)
```

End Sub

```
Private Sub BtnBrowseSavers_Click()
CommonDialog1.ShowOpen
TxtScreenSaver.Text = CommonDialog1.filename
End Sub
```

```
Private Sub BtnCancelIt_Click()
    Unload Me
    'frmConfig.Hide
End Sub
```

```
Private Sub BtnHelpT3_Click()
    SendKeys "{F1}"
End Sub
```

```
Private Sub BtnHelpT4_Click()
    SendKeys "{F1}"
End Sub
```

```
Private Sub BtnOK_Click()
    Call CfgUpdate
    Unload Me
    'frmConfig.Hide
End Sub
```

```
Private Sub Check1_Click()

End Sub
```

```
Private Sub ChkT0Compress_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
ChkT0Compress.SetFocus
    If Button = vbRightButton Then
        SendKeys "{F1}"
    End If
End Sub
```

```
Private Sub ChkT0ConfirmEncrypt_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
ChkT0ConfirmEncrypt.SetFocus
    If Button = vbRightButton Then
        SendKeys "{F1}"
    End If
End Sub
```

frmConfig - 4

End If
End Sub

```
Private Sub ChkT0ConfirmRO_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
    ChkT0ConfirmRO.SetFocus
    If Button = vbRightButton Then
        SendKeys "{F1}"
    End If
End Sub
```

```
Private Sub ChkT0Shred_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
    ChkT0Shred.SetFocus
    If Button = vbRightButton Then
        SendKeys "{F1}"
    End If
End Sub
```

```
Private Sub ChkT3Authenticate_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
    ChkT3Authenticate.SetFocus
    If Button = vbRightButton Then
        SendKeys "{F1}"
    End If
End Sub
```

```
Private Sub ChkT3LaunchApp_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
    ChkT3LaunchApp.SetFocus
    If Button = vbRightButton Then
        SendKeys "{F1}"
    End If
End Sub
```

```
Private Sub Command1_Click()
    SendKeys "{F1}"
End Sub
```

```
Private Sub Command2_Click()
    SendKeys "{F1}"
End Sub
```

```
Private Sub Form_Load()
    ChkT0Compress.Value = zzCompress
    ChkT0ConfirmEncrypt.Value = zzConfirmEncrypt
    ChkT0ConfirmRO.Value = zzConfirmRO
    ChkT0Shred.Value = zzShred
    ChkT3Authenticate.Value = zzAuthenticate
    ChkT3Authenticate.Enabled = False
    OptT1LongFileName.Enabled = True
    OptT1LongFileExt.Enabled = True
    OptT1ShortFileName.Enabled = True
    OptT1ShortFileExt.Enabled = True
    OptT1NoNameChange.Enabled = True
```

```
OptT1LongFileName.Value = zzLongFileName
OptT1LongFileExt.Value = zzLongFileExt
OptT1ShortFileName.Value = zzShortFileName
OptT1ShortFileExt.Value = zzShortFileExt
OptT1NoNameChange.Value = zzNoNameChange
OptT1CustomExt.Value = zzCustomExt
OptT2NetLogIn.Value = zzNetLogIn
OptT2NetLogIn.Enabled = True
OptT2ScreenSaver.Value = zzScreenSaver
OptT2ScreenSaver.Enabled = False
OptT2NoStandby.Value = zzNoStandby
```

```
If OptT2NoStandby.Value = True Then
```

frmConfig - 5

```
    TxtTimeOut.Enabled = False
End If
```

```
OptT2ExitSystem.Value = zzExitSystem
C    ?ExitSystem.Enabled = False
OptT2ShutDown.Value = zzShutDown
OptT2ShutDown.Enabled = True
OptT2StartSaver.Value = zzStartSaver
OptT2StartSaver.Enabled = True
OptT2IgnorSC.Value = zzIgnorSC
```

```
TxtTimeOut.Text = zzTxtTimeOut
TxtTimeOut.Enabled = False
'TxtServerName.Text = zzTxtServerName
'TxtServerName.Enabled = False
TxtCustomExt.Text = zzTxtCustomExt
TxtScreenSaver.Enabled = False
TxtScreenSaver.Text = zzTxtScreenSaverName
```

```
Label5.Caption = zzFileMode
'T5DisableDos.Enabled = False
'T5disableIcons.Enabled = False
'T5DisableTray.Enabled = False
'T5DisableStart.Enabled = False
'T5DisableTaskBar.Enabled = False
'T3ChkDisableFloppy.Enabled = False
'T3ChkdisableModem.Enabled = False
'T3chkDisableRemovable.Enabled = False
```

```
If OptT1CustomExt = True Then
    TxtCustomExt.Enabled = True
Else
    TxtCustomExt.Enabled = False
End If
```

End Sub

```
Private Sub Option2_Click()
```

End Sub

```
Private Sub OptT1CustomExt_Click()
If OptT1CustomExt.Value Then
    TxtCustomExt.Enabled = True
Else
    TxtCustomExt.Enabled = False
End If
```

End Sub

```
Private Sub OptT1LongFileExt_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
OptT1LongFileExt.SetFocus
    If Button = vbRightButton Then
        SendKeys "{F1}"
    End If
End Sub
```

```
Private Sub OptT1LongFileName_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
OptT1LongFileName.SetFocus
    If Button = vbRightButton Then
        SendKeys "{F1}"
    End If
```

frmConfig - 6

End Sub

```
Private Sub OptT1NoNameChange_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
OptT1NoNameChange.SetFocus
```

```
    If Button = vbRightButton Then
        SendKeys "{F1}"
```

```
    End If
```

End Sub

```
Private Sub OptT1ShortFileExt_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
OptT1ShortFileExt.SetFocus
```

```
    If Button = vbRightButton Then
        SendKeys "{F1}"
```

```
    End If
```

End Sub

```
Private Sub OptT1ShortFileName_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
OptT1ShortFileName.SetFocus
```

```
    If Button = vbRightButton Then
        SendKeys "{F1}"
```

```
    End If
```

End Sub

```
Private Sub OptT2ExitSystem_Click()
```

```
TxtTimeOut.Enabled = True
```

End Sub

```
Private Sub OptT2ExitSystem_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
OptT2ExitSystem.SetFocus
```

```
    If Button = vbRightButton Then
        SendKeys "{F1}"
```

```
    End If
```

End Sub

```
Private Sub OptT2IgnorSC_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
OptT2IgnorSC.SetFocus
```

```
    If Button = vbRightButton Then
        SendKeys "{F1}"
```

```
    End If
```

End Sub

```
Private Sub OptT2NetLogIn_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
OptT2NetLogIn.SetFocus
```

```
    If Button = vbRightButton Then
        SendKeys "{F1}"
```

```
    End If
```

End Sub

```
Private Sub OptT2NoStandby_Click()
```

```
TxtTimeOut.Enabled = False
```

End Sub

```
Private Sub OptT2NoStandby_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
OptT2NoStandby.SetFocus
```

```
    If Button = vbRightButton Then
        SendKeys "{F1}"
```

```
    End If
```

End Sub

```
Private Sub OptT2ScreenSaver_Click()
```

```
TxtTimeOut.Enabled = True
```

End Sub

```

Private Sub OptT2ScreenSaver_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
OptT2ScreenSaver.SetFocus
    If Button = vbRightButton Then
        SendKeys "{F1}"
    End If
End Sub

Private Sub OptT2ShutDown_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
OptT2ShutDown.SetFocus
    If Button = vbRightButton Then
        SendKeys "{F1}"
    End If
End Sub

Private Sub OptT2StartSaver_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
OptT2StartSaver.SetFocus
    If Button = vbRightButton Then
        SendKeys "{F1}"
    End If
End Sub

Private Sub TxtCustomExt_Change()
If Len(TxtCustomExt.Text) > 3 Then
    MsgBox ("The file extension is limited to 3 characters")
    TxtCustomExt.Text = ""
    Exit Sub
End If

If Left(TxtCustomExt.Text, 1) = "." Then
    TxtCustomExt.Text = Right(TxtCustomExt.Text, Len(TxtCustomExt.Text) - 1)
    TxtCustomExt.Refresh
    Exit Sub
End If

zzTxtCustomExt = TxtCustomExt.Text
End Sub

```


frmDisplayCard - 1

```
Private Sub BtnOK_Click()  
    Unload Me  
End Sub  
  
Private Sub Form_Load()  
  
    Dim cNames As Collection  
    Dim n As Integer  
  
    Set cNames = oServer.KeyNames  
  
    If Not oAccess.ValidateUser("z1001941") Then  
        MsgBox "Smart Card Server Refused Access"  
        Unload Me  
    End If  
  
    For n = 1 To cNames.Count  
        lstKeys.AddItem cNames.Item(n)  
    Next  
    TxtCardID.Text = oAccess.sCardUserID  
  
End Sub
```

}

```
frmDisplayStatus - 1
```

```
Private Sub mnuFileClose_Click()  
    frmDisplayStatus.Hide  
End Sub
```

```
Private Sub Command1_Click()  
    Dim Counter As Integer  
    Dim Workarea(200) As String  
    Dim NumFiles As Integer  
  
    NumFiles = FileList1.ListCount
```

```
    ProgressBar1.Min = LBound(Workarea)  
    ProgressBar1.Max = UBound(Workarea)  
    ProgressBar1.Visible = True
```

```
'Set the Progress's Value to Min.  
    ProgressBar1.Value = ProgressBar1.Min
```

```
'Loop through the array.  
    For Counter = LBound(Workarea) To UBound(Workarea)  
        'Set initial values for each item in the array.  
        Workarea(Counter) = "Initial value" & Counter  
        ProgressBar1.Value = Counter
```

```
    Next Counter  
    ProgressBar1.Visible = False  
    ProgressBar1.Value = ProgressBar1.Min  
End Sub
```

```
Private Sub Form_Load()  
    '    ProgressBar1.Align = vbAlignBottom  
    ProgressBar1.Visible = True  
    Command1.Caption = "Start"  
End Sub
```

frmFileSelect - 1

```
Dim prepreturn As Long
Dim decreturn As Long
Dim encreturn As Long
Dim keystring As String
Dim k_ID As String * 128
Dim fileToEncrypt As String
Dim SetOK As Long
Dim sFileName As String
Dim bFileAdd As Boolean
Dim nAction As Long
Dim scKeyName As String * 16
Dim scKeyValue As String * 8
Dim sKName As String
Dim sKValue As String
```

```
Private Sub ProgBar()
Dim Counter As Integer
    Dim Workarea(250) As String
    Dim NumFiles As Integer
```

```
NumFiles = frmDisplayStatus.FileList1.ListCount
```

```
frmDisplayStatus.ProgressBar1.Min = LBound(Workarea)
frmDisplayStatus.ProgressBar1.Max = UBound(Workarea)
frmDisplayStatus.ProgressBar1.Visible = True
```

```
'Set the Progress's Value to Min.
frmDisplayStatus.ProgressBar1.Value = frmDisplayStatus.ProgressBar1.Min
```

```
'Loop through the array.
For Counter = LBound(Workarea) To UBound(Workarea)
    'Set initial values for each item in the array.
    Workarea(Counter) = "Initial value" & Counter
    frmDisplayStatus.ProgressBar1.Value = Counter
```

```
Next Counter
frmDisplayStatus.ProgressBar1.Visible = False
frmDisplayStatus.ProgressBar1.Value = frmDisplayStatus.ProgressBar1.Min
```

End Sub

```
Private Sub BtnDecryptIt_Click()
Dim loop_ctr As Integer
Dim InFile As String
Dim OutFile As String * 128
```

```
If List1.ListCount = 0 Then
    SetOK = MsgBox("There are no files selected to encrypt", vbInformation, "IntelliGard Smart Suite")
    Exit Sub
End If
```

```
'If Not oAccess.ValidateUser("z1001941") Then

    'MsgBox "Smart Card Server Refused Access"
    ' Unload Me

' End If
```

```
loop_ctr = List1.ListCount
'Screen.MousePointer = vbCustom
```

```
Do Until loop_ctr = 0

    InFile = List1.List(0)
    OutFile = "+"
```

```
SetOK = SetFileAttributes(InFile, FILE_ATTRIBUTE_ARCHIVE)
```

```
prepreturn = PrepDec(InFile, k_ID)
```

```
zzKeySelected = sTrimNulls(k_ID)
```

```
sKeyValue = oServer.KeyValue(zzKeySelected)
```

```
If sKeyValue = "empty" Then
```

```
    GoSub NoKeyValue
```

```
Else
```

```
    sKeyValue = sTrimNulls(sKeyValue)
```

```
End If
```

```
Select Case prepreturn
```

```
    Case 1
```

```
        Screen.MousePointer = vbCustom
```

```
        decreturn = DecFile(OutFile, sKeyValue)
```

```
    Case -1
```

```
        GoSub WrongVer
```

```
    Case 0
```

```
        GoSub NotEncrypt
```

```
End Select
```

```
If frmDisplayStatus.OptProgBar Then
```

```
    Call ProgBar
```

```
End If
```

```
' Remove item from frmDisplayStatus
```

```
If decreturn = 1 Then
```

```
    List1.RemoveItem 0
```

```
    frmDisplayStatus.FileList1.RemoveItem 0
```

```
    LogDecrypt (InFile)
```

```
Else: GoSub DecError
```

```
End If
```

```
loop_ctr = loop_ctr - 1
```

```
List1.Refresh
```

```
frmDisplayStatus.FileList1.Refresh
```

```
Loop
```

```
File1.Refresh
```

```
Screen.MousePointer = vbDefault
```

```
Exit Sub
```

```
DecError:
```

```
    nAction = MsgBox(InFileName & Chr(10) & "This File can not be decrypted, it may be corrupt,  
would you like to continue?", vbExclamation + vbYesNo, "IntelliGard Smart Suite")
```

```
    If nAction = 6 Then
```

```
        List1.RemoveItem 0
```

```
        frmDisplayStatus.FileList1.RemoveItem 0
```

```
        File1.Refresh
```

```
    Else
```

```
        List1.Clear
```

```
        frmDisplayStatus.FileList1.Clear
```

```
        Screen.MousePointer = vbDefault
```

```
        Exit Sub
```

```
    End If
```

```
Screen.MousePointer = vbDefault
```

```
Return
```

```
WrongVer:
```

frmFileSelect - 3

```
nAction = MsgBox(InFileName & Chr(10) & "This File can not be decrypted, it was encrypted with an old version, would you like to continue?", vbExclamation + vbYesNo, "IntelliGard Smart Suite")
)
If nAction = 6 Then
    List1.RemoveItem 0
    frmDisplayStatus.FileList1.RemoveItem 0
    File1.Refresh
Else
    List1.Clear
    frmDisplayStatus.FileList1.Clear
    Screen.MousePointer = vbDefault
    Exit Sub
End If

Screen.MousePointer = vbDefault
Return

NotEncrypt:
nAction = MsgBox(InFileName & Chr(10) & "This File has not been encrypted, would you like to continue?", vbExclamation + vbYesNo, "IntelliGard Smart Suite")
If nAction = 6 Then
    List1.RemoveItem 0
    frmDisplayStatus.FileList1.RemoveItem 0
    File1.Refresh
Else
    List1.Clear
    frmDisplayStatus.FileList1.Clear
    Screen.MousePointer = vbDefault
    Exit Sub
End If

Screen.MousePointer = vbDefault
Return

If /Value:
nAction = MsgBox(InFileName & Chr(10) & "Key access denied, you can not decrypt this file, would you like to continue?", vbExclamation + vbYesNo, "IntelliGard Smart Suite")
If nAction = 6 Then
    List1.RemoveItem 0
    frmDisplayStatus.FileList1.RemoveItem 0
    File1.Refresh
Else
    List1.Clear
    frmDisplayStatus.FileList1.Clear
    Screen.MousePointer = vbDefault
    Exit Sub
End If

Screen.MousePointer = vbDefault
Return

End Sub

Private Sub BtnDecryptIt_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
    BtnDecryptIt.SetFocus
    If Button = vbRightButton Then
        SendKeys "{F1}"
    End If
End Sub

End Sub

Private Sub BtnDeselect_Click(Index As Integer)
    List1.Clear
    frmDisplayStatus.FileList1.Clear
End Sub

Private Sub BtnEncryptCopy_Click()
    If List1.ListCount = 0 Then
        SetOK = MsgBox("There are no files selected to encrypt", vbInformation, "IntelliGard Smart Suite")
    End If
End Sub
```

frmFileSelect - 4

```
Exit Sub
End If

If zzFileMode = 6 Then
    MsgBox "You Can't Make a Copy With The File Naming Mode Currently In Use. Please Change Pr
e  nces"
    Exit Sub
End If

' If Not oAccess.ValidateUser("z1001941") Then

    ' MsgBox "Smart Card Server Refused Access"
    ' Unload Me

' End If

If Not oServer.DisplayReturnKey(scKeyName, scKeyValue) Then
    SetOK = MsgBox("Key Access Denied. Please Contact Your System Administrator", vbExclamation
+ vbOKOnly, "IntelliGard Smart Suite")
    Exit Sub
End If

sKName = Trim(scKeyName)
sKValue = Trim(scKeyValue)

If sKName = "cancel" Or sKValue = "empty" Then
    nError = MsgBox("Encryption Cancelled by User", vbExclamation + vbOKOnly, "IntelliGard Smar
t Suite")
    List1.Clear
    frmDisplayStatus.FileList1.Clear
    Exit Sub
End If

Screen.MousePointer = vbCustom

Dim loop_ctr As Integer
PassCtrl = DoEvents
loop_ctr = List1.ListCount

Do Until loop_ctr = 0
    InFileName = List1.List(0)
    SetOK = SetFileAttributes(InFileName, FILE_ATTRIBUTE_ARCHIVE)

    Call EncNamePrep

    encreturn = EncFileNoDestroy(InFileName, OutFileName, sKName, sKValue)

    If frmDisplayStatus.OptProgBar Then
        Call ProgBar
    End If

    ' Remove item from frmDisplayStatus

    If encreturn = 1 Then
        List1.RemoveItem 0
        frmDisplayStatus.FileList1.RemoveItem 0
        SetOK = SetFileAttributes(OutFileName, FILE_ATTRIBUTE_READONLY)
        Call LogEncrypt(OutFileName, sKName)

    Else: GoSub EncError
    End If
    loop_ctr = loop_ctr - 1
    List1.Refresh
    frmDisplayStatus.FileList1.Refresh
Loop
File1.Refresh
```

frmFileSelect - 5

Screen.MousePointer = vbDefault

Exit Sub

Error:
nAction = MsgBox(InFileName & Chr(10) & "This File is already encrypted, would you like to
continue?", vbExclamation + vbYesNo, "IntelliGard Smart Suite")

If nAction = 6 Then
List1.RemoveItem 0
frmDisplayStatus.FileList1.RemoveItem 0
File1.Refresh

Else
List1.Clear
frmDisplayStatus.FileList1.Clear
Screen.MousePointer = vbDefault
Exit Sub
End If

Screen.MousePointer = vbDefault
Return

End Sub

Private Sub BtnEncryptMail_Click()

Dim ReCrypt As Integer
Dim RetVal1 As Integer
Dim RetVal2 As Integer
Dim OutFile As String * 128
Dim InFile As String
Dim PassCtrl As Long
Dim loop_ctr As Integer

If List1.ListCount = 0 Then
SetOK = MsgBox("There are no files selected to encrypt", vbInformation, "IntelliGard Smart S
uite")

Exit Sub
End If
loop_ctr = List1.ListCount
Screen.MousePointer = vbCustom

Do Until loop_ctr = 0
InFileName = List1.List(0)
SetOK = SetFileAttributes(InFileName, FILE_ATTRIBUTE_ARCHIVE)
OutFile = "+"
k_ID = "Demo Key"
keystring = "12345678"

RetVal1 = PrepDec(InFileName, k_ID)

If RetVal1 <> 1 Then
MsgBox "The File is Not Encrypted, Simply Select The File and Hit the Encrypt Key"
Screen.MousePointer = vbDefault
Exit Sub
End If

RetVal2 = DecFileNoDestroy(OutFile, keystring)

InFile = Trim(OutFile)

EncName = Left(InFile, Len(InFile) - 4) + "maz"
k_ID = "Mail Key"

encreturn = EncFile(InFile, EncName, k_ID, keystring)

If encreturn = 1 Then
List1.RemoveItem 0
frmDisplayStatus.FileList1.RemoveItem 0
SetOK = SetFileAttributes(OutFileName, FILE_ATTRIBUTE_READONLY)

frmFileSelect - 6

Call LogEncrypt(OutFileName, k_ID)

Else: GoSub EncError

End If

loop_ctr = loop_ctr - 1

List1.Refresh

frmDisplayStatus.FileList1.Refresh

Loop

File1.Refresh

Screen.MousePointer = vbDefault

Exit Sub

EncError:

nAction = MsgBox(InFileName & Chr(10) & "This File can not be encrypted, it may be corrupt
, would you like to continue?", vbExclamation + vbYesNo, "IntelliGard Smart Suite")

If nAction = 6 Then

List1.RemoveItem 0

frmDisplayStatus.FileList1.RemoveItem 0

File1.Refresh

Else

List1.Clear

frmDisplayStatus.FileList1.Clear

Screen.MousePointer = vbDefault

Exit Sub

End If

Screen.MousePointer = vbDefault

Return

End Sub

Private Sub BtnEncryptIt_Click(Index As Integer)

If List1.ListCount = 0 Then

SetOK = MsgBox("There are no files selected to encrypt", vbInformation, "IntelliGard Smart Suite")

Exit Sub

End If

If Not oAccess.ValidateUser("z1001941") Then

MsgBox "Smart Card Server Refused Access"

Unload Me

End If

If Not oServer.DisplayReturnKey(scKeyName, scKeyValue) Then

SetOK = MsgBox("Key Access Denied. Please Contact Your System Administrator", vbExclamation + vbOKOnly, "IntelliGard Smart Suite")

Exit Sub

End If

sKName = Trim(scKeyName)

sKValue = Trim(scKeyValue)

If sKName = "cancel" Or sKValue = "empty" Then

nError = MsgBox("Encryption Cancelled by User", vbExclamation + vbOKOnly, "IntelliGard Smart Suite")

List1.Clear

frmDisplayStatus.FileList1.Clear

Exit Sub

End If

Screen.MousePointer = vbCustom

Dim loop_ctr As Integer

PassCtrl = DoEvents

frmFileSelect - 7

loop_ctr = List1.ListCount

Do Until loop_ctr = 0
InFileName = List1.List(0)

SetOK = SetFileAttributes(InFileName, FILE_ATTRIBUTE_ARCHIVE)

Call EncNamePrep

If sKName = "M: Mail" Then
OutFileName = Left(InFileName, Len(InFileName) - 4) & ".maz"
End If

If zzFileMode = 6 Then
encreturn = EncFileNoDestroy(InFileName, OutFileName, sKName, sKValue)
Else
encreturn = EncFile(InFileName, OutFileName, sKName, sKValue)
End If

If frmDisplayStatus.OptProgBar Then
Call ProgBar
End If

' Remove item from frmDisplayStatus

If encreturn = 1 Then
List1.RemoveItem 0
frmDisplayStatus.FileList1.RemoveItem 0
SetOK = SetFileAttributes(OutFileName, FILE_ATTRIBUTE_READONLY)
Call LogEncrypt(OutFileName, sKName)

Else: GoSub EncError
End If

loop_ctr = loop_ctr - 1
List1.Refresh
frmDisplayStatus.FileList1.Refresh

Loop
File1.Refresh
Screen.MousePointer = vbDefault

Exit Sub

EncError:

nAction = MsgBox(InFileName & Chr(10) & "This File is already encrypted, would you like to
continue?", vbInformation + vbYesNo, "IntelliGard Smart Suite")

If nAction = 6 Then
List1.RemoveItem 0
frmDisplayStatus.FileList1.RemoveItem 0
File1.Refresh

Else
List1.Clear
frmDisplayStatus.FileList1.Clear
Screen.MousePointer = vbDefault
Exit Sub
End If

Screen.MousePointer = vbDefault
Return

End Sub

Private Sub BtnEncryptIt_MouseDown(Index As Integer, Button As Integer, Shift As Integer, X As Single, Y As Single)

BtnEncryptIt(0).SetFocus
If Button = vbRightButton Then

frmFileSelect - 8

```
        SendKeys "{F1}"
    End If
```

End Sub

```
Private Sub BtnSelectKey_Click()
    frmSelectKey.Show
End Sub
```

```
Private Sub BtnHelp_Click()
    SendKeys "{F1}"
End Sub
```

```
Private Sub CancelIt_Click()
    If chkSaveDefault.Value = 1 Then
        UpdateDefaults
    End If
```

```
    List1.Clear
    frmDisplayStatus.FileList1.Clear
    frmFileSelect.Hide
```

End Sub

```
Private Sub Check1_Click()
```

End Sub

```
Private Sub ChkShowStatus_Click()
```

```
    If ChkShowStatus Then
        frmDisplayStatus.Show
```

```
    Else
        frmDisplayStatus.Hide
```

```
    End If
```

```
End Sub
```

```
Private Sub Dir1_Change()
```

```
    File1.Path = Dir1.Path
```

End Sub

```
Private Sub Drive1_Change()
```

```
    Dir1.Path = Drive1.Drive
```

End Sub

```
Private Sub File1_DblClick()
```

```
    Dim sSelectedFile As String
```

```
    If Right(Dir1.Path, 1) = "\" Then
        sSelectedFile = Dir1.Path & File1
```

```
    Else
        sSelectedFile = Dir1.Path + "\" + File1
```

```
    End If
```

```
    bFileAdd = FileTypeWarning(sSelectedFile)
```

```
    If bFileAdd Then
```

```
        List1.AddItem sSelectedFile
```

```
        frmDisplayStatus.FileList1.AddItem sSelectedFile
```

```
    End If
```

Sub

```
Private Sub Form_Load()
```

```
    'If zzDefaultPath <> "1" Then
```

```
    'File1.Path = zzDefaultPath
```

```
    'Dir1.Path = zzDefaultPath
```

```
    'End If
```

frmFileSelect - 9

Dim sIconName

sIconName = App.Path & "\dave.ico"

Screen.MouseIcon = LoadPicture(sIconName)

File1.Refresh
End Sub

Private Sub Form_Unload(Cancel As Integer)

Unload Me
End Sub

Private Sub List1_DblClick()

DelTarget = List1.ListIndex
List1.RemoveItem DelTarget
frmDisplayStatus.FileList1.RemoveItem DelTarget

End Sub

Private Sub SelectAll_Click()

Dim i
For i = 0 To File1.ListCount - 1

If File1.Selected(i) Then
If Right(Dir1.Path, 1) = "\" Then
sFileName = Dir1.Path & File1.List(i)
Else
sFileName = Dir1.Path + "\" + File1.List(i)
End If
bAddFile = FileTypeWarning(sFileName)

If bAddFile Then
List1.AddItem sFileName
frmDisplayStatus.FileList1.AddItem sFileName
End If

End If

Next i

End Sub

Private Sub UpdateDefaults()

Dim zApp As String
Dim zstring As String
Dim zKeyName As String
Dim zDefault As String
Dim zReturn As String
Dim zSize As Long
Dim RetOK As Long

zApp = "General"
zDefault = "0"
zSize = 32

zzDefaultPath = File1.Path
zstring = File1.Path
zKeyName = "zzDefaultPath"
RetOK = WritePrivateProfileString(zApp, zKeyName, zstring, CfgFile)
End Sub

frmIgardMain - 1

```
Private Declare Function GetUserName Lib "advapi32.dll" Alias "GetUserNameA" (ByVal lpBuffer As String, nSize As Long) As Long
Private Declare Function GetComputerName Lib "kernel32" Alias "GetComputerNameA" (ByVal lpBuffer As String, nSize As Long) As Long
```

```
Private Sub LogOutEntry()
    Open UserLog For Append As #1
    Print #1, "Smart Suite"; " "; CurrentUserName; " "; "- Out"; " "; Date; Time(); Chr$(13);
    Chr$(10);
    Close #1
End Sub
```

```
Private Sub LogInEntry()
    Open UserLog For Append As #1
    Print #1, "Smart Suite"; " "; CurrentUserName; " "; "- In"; " "; Date; Time(); Chr$(13);
    Chr$(10);
    Close #1
End Sub
```

```
Private Sub Command1_Click()
    Dim ByeOK As Long
    ByeOK = ExitWindowsEx(EWX_REBOOT, 0)
    Call LogOutEntry
End Sub
```

```
Private Sub BtnConfig_Click()
    frmConfig.Show
End Sub
```

```
Private Sub BtnDecrypt_Click()
    frmFileSelect.Show vbModal
End Sub
```

```
Private Sub BtnDecrypt_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
    BtnDecrypt.SetFocus
    If Button = vbRightButton Then
        SendKeys "{F1}"
    End If
End Sub
```

```
Private Sub BtnEncrypt_Click()
    frmFileSelect.Show vbModal
End Sub
```

```
Private Sub BtnEncrypt_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
    BtnEncrypt.SetFocus
    If Button = vbRightButton Then
        SendKeys "{F1}"
    End If
End Sub
```

```
Private Sub BtnHelp_Click()
    CommonDialog1.HelpFile = "E Series Help.HLP"
    CommonDialog1.HelpCommand = cdlHelpContents
    CommonDialog1.ShowHelp ' Display Visual Basic Help contents topic.
    SendKeys "{F1}"
End Sub
```

```
Private Sub BtnLogin_Click()
    If oServer.DisplayKeys Then
        End If
End Sub
```

```
End Sub
```

```
Private Sub BtnPassword_Click()
    frmPassChange.Show vbModal
End Sub
```

```
Private Sub Command6_Click()
```

End Sub

```
Private Sub BtnSysInfo_Click()
    ' Call StartSysInfo
End Sub
```

```
Private Sub Command2_Click()
    Dim ByeOK As Long
    ByeOK = ExitWindowsEx(EWX_LOGOFF, 0)
    Unload Me
End Sub
```

```
Private Sub Form_Load()
    Dim CurrentComputerName As String
    SystemStarted = True
    mnviewcontrol.Checked = True
    mnviewtools.Checked = False

    YNStyle = vbYesNo + vbCritical + vbDefaultButton2
    ' Get the User Name From Network

    Dim sBuffer As String
    Dim lSize As Long

    sBuffer = Space$(255)
    lSize = Len(sBuffer)
    Call GetUserName(sBuffer, lSize)
    If lSize > 0 Then
        CurrentUserName = Left$(sBuffer, lSize)
    Else
        CurrentUserName = vbNullString
    End If

    sBuffer = Space$(255)
    lSize = Len(sBuffer)
    Call GetComputerName(sBuffer, lSize)
    If lSize > 0 Then
        CurrentComputerName = LCase(Left$(sBuffer, lSize))
    Else
        CurrentComputerName = vbNullString
    End If

    TxtPanelUserName.Text = CurrentUserName + " is logged in"

    TxtComputerName.Text = CurrentComputerName
```

```
' Initialize Control Panel Off
BtnEncrypt.Visible = True
BtnDecrypt.Visible = True
BtnLogin.Visible = True
BtnPassword.Visible = True
BtnHelp.Visible = True
Frame1.Visible = True
Frame2.Visible = True
Frame3.Visible = True
List1.Visible = True
TxtPanelUserName.Visible = True
```

```
' Initialize SC Key List
List1.Enabled = False
Dim cNames As Collection
Dim n As Integer

Set cNames = oServer.KeyNames
```

frmIgardMain - 3

```
For n = 1 To cNames.Count
    List1.AddItem cNames.Item(n)
Next n
Call LogInEntry

If zzShred Then
    mnuFileShred.Enabled = True
Else
    mnuFileShred.Enabled = False
End If

End Sub

Private Sub mnuChgPass_Click()
    frmPassChange.Show vbModal
End Sub

Private Sub mnuDisplaySmartCard_Click()
    frmDisplayCard.Show
End Sub

Private Sub mnuDisplayStatus_Click()
    frmDisplayStatus.Show
End Sub

Private Sub mnuExit_Click()
    Unload Me 'kill system
End Sub

Private Sub mnuFileClose_Click()
    frmIgardMain.Hide
End Sub

Private Sub mnuFileDecrypt_Click()
    frmFileSelect.Show vbModal
End Sub

Private Sub mnuFileEncrypt_Click()
    frmFileSelect.Show vbModal
End Sub

Private Sub mnuFileExit_Click()
    Call LogOutEntry
End Sub

End Sub

Private Sub mnuFileLogin_Click()
    frmLogin.Show
End Sub

Private Sub mnuFileMinimize_Click()
    Dim CloseOK&
    CloseOK = CloseWindow(hwnd)
End Sub

Private Sub mnuFilePreferences_Click()
    frmConfig.Show vbModal
End Sub

Private Sub mnuFileShred_Click()
    frmShred.Show vbModal
End Sub

Private Sub mnuFileStatus_Click()
    frmDisplayStatus.Show vbModal
End Sub
```

frmIgardMain - 4

```
Private Sub mnuHelpAbout_Click()  
    frmAbout.Show vbModal  
End Sub  
  
Private Sub mnuHelpContents_Click()  
    'CommonDialog1.HelpFile = "E Series Help.HLP"  
    'CommonDialog1.HelpCommand = cdlHelpContents  
    'CommonDialog1.ShowHelp ' Display Visual Basic Help contents topic.  
    SendKeys "{F1}"  
End Sub  
  
Private Sub mnuOpenIntelligard_Click()  
    frmIgardMain.Show  
End Sub  
  
Private Sub mnuReLogin_Click()  
    frmLogin.Show  
End Sub  
  
Private Sub mnuSACryptolog_Click()  
    'LogFileType = 2  
    'Load frmViewer  
    'frmViewer.Show  
Dim CryptEx As String  
Dim RetOK As Long  
  
CryptEx = "notepad.exe " & CryptoLog  
RetOK = Shell(CryptEx, 1)  
  
End Sub  
  
Private Sub mnuSAPreferences_Click()  
    LogFileType = 3  
    Load frmViewer  
    frmViewer.Show  
End Sub  
  
Private Sub mnuSAuserlog_Click()  
    ' LogFileType = 1  
    'Load frmViewer  
    'frmViewer.Show  
Dim LogEx As String  
Dim RetOK As Long  
  
LogEx = "notepad.exe " & UserLog  
RetOK = Shell(LogEx, 1)  
  
End Sub  
  
Private Sub mnuSCDisplay_Click()  
    If oServer.DisplayKeys Then  
        End If  
  
End Sub  
  
Private Sub mnuSCReLogin_Click()  
    frmLogin.Show vbModal  
End Sub  
  
Private Sub mnuserverlog_Click()  
Dim LogEx As String  
Dim RetOK As Long  
(  
LogEx = "notepad.exe " & UserLog  
RetOK = Shell(LogEx, 1)  
  
End Sub  
  
Private Sub mnuupdateuser_Click()
```

frmIgardMain - 5

Call UpdateUserType
frmIgardMain.Refresh

End Sub

Private Sub mnuviewcontrol_Click()
mnuviewcontrol.Checked = Not mnuviewcontrol.Checked

If mnuviewcontrol.Checked Then
 BtnEncrypt.Visible = True
 BtnDecrypt.Visible = True
 BtnLogin.Visible = True
 BtnPassword.Visible = True
 BtnHelp.Visible = True
 Frame1.Visible = True
 Frame2.Visible = True
 Frame3.Visible = True
 List1.Visible = True
 TxtPanelUserName.Visible = True

Else
 BtnEncrypt.Visible = False
 BtnDecrypt.Visible = False
 BtnLogin.Visible = False
 BtnPassword.Visible = False
 BtnHelp.Visible = False
 Frame1.Visible = False
 Frame2.Visible = False
 Frame3.Visible = False
 List1.Visible = False
 TxtPanelUserName.Visible = False

End If
End Sub

Private Sub mnuviewtools_Click()
mnuviewtools.Checked = Not mnuviewtools.Checked
 If mnuviewtools.Checked Then
 Toolbar1.Visible = True
 Line1.Visible = True
 Else
 Toolbar1.Visible = False
 Line1.Visible = False
 End If

End Sub

Private Sub Toolbar1_ButtonClick(ByVal Button As ComctlLib.Button)
Select Case Button.Key
 Case Is = "encrypt"
 frmFileSelect.Show vbModal
 Case Is = "decrypt"
 frmFileSelect.Show vbModal
 Case Is = "displaycard"
 frmDisplayCard.Show
 Case Is = "preferences"
 frmConfig.Show vbModal
 Case Is = "help me"
 SendKeys "{F1}"
 Case Is = "exit"
 Call LogOutEntry
End

 Case Else
 ' If any other button is pressed
End Select
End Sub

frmPassChange - 1

```
Private Sub BtnCancelIt_Click()  
    Unload frmPassChange  
End Sub
```

```
Private Sub BtnChangeIt_Click()  
    Dim sOldPass As String  
    Dim sNewPass As String
```

```
If Trim(TxtNewPass.Text) = Trim(TxtConfirm.Text) Then  
    sOldPass = Trim(TxtOldPass.Text)  
    sNewPass = Trim(TxtNewPass.Text)  
    MsgBox "Please Make Sure Your Smart Card is Inserted, This Will Take a Moment..."  
    If oServer.ChangePassword(sOldPass, sNewPass) Then  
        MsgBox "Password Sucessfully Changed, Hit OK to Continue"  
        Unload frmPassChange  
    Else  
        MsgBox "Password Not Changed, Must Be Changed By System Administrator"  
        Unload frmPassChange  
    End If
```

```
Else  
    MsgBox "Please Re-Confirm Your New Password and Hit Change Again"  
End If  
  
End Sub
```

frmShred - 1

```
Dim FileToShred As String
Private Sub CancelIt_Click()
    Text1.Text = " "
    frmShred.Hide
End Sub
```

```
Private Sub Dir1_Change()
    File1.Path = Dir1.Path
End Sub
```

```
Private Sub Drive1_Change()
    Dir1.Path = Drive1.Drive
End Sub
```

```
Private Sub File1_DblClick()
```

```
    Text1.Text = Dir1.Path + "\" + File1
    FileToShred = Text1.Text
    If Not FileTypeWarning(FileToShred) Then
        Text1.Text = " "
        Text1.Refresh
        Exit Sub
    End If
```

```
End Sub
```

```
Private Sub ShredIt_Click()
```

```
    Dim ShredOK As Integer
    Dim shredded As Long
    Dim Fattr As Long
```

```
    Fattr = GetFileAttributes(FileToShred)
```

```
    If Fattr = FILE_ATTRIBUTE_READONLY Or Fattr = FILE_ATTRIBUTE_SYSTEM Then
```

```
        ShredOK = MsgBox("This File is a Read Only File, Are You Sure You Want to Destroy It?", YNS
tyle)
```

```
    Else
```

```
        ShredOK = MsgBox("The File You Are About to Shred Will Be Permanently Removed!, Continue?", „
```

```
YNStyle)
```

```
    End If
```

```
    If ShredOK = 6 Then
```

```
        Fattr = SetFileAttributes(FileToShred, FILE_ATTRIBUTE_ARCHIVE)
```

```
        If oAccess.GetSCTime Then
```

```
            End If
```

```
            shredded = Shred(FileToShred)
```

```
            Text1.Text = " "
```

```
            File1.Refresh
```

```
    Else
```

```
        Text1.Text = " "
```

```
        File1.Refresh
```

```
    End If
```

```
    If oAccess.GetSCDate Then
```

```
        End If
```

```
End Sub
```

:

(



logfiles - 1

Option Explicit

Public Sub LogEncrypt(filename As String, keyname As String)

Open CryptoLog For Append As #3

Print #3, "Encrypt: "; CurrentUserName; " - Console"; " - "; Trim(filename); " "; Trim(keyname); " - "; Date; Time(); Chr\$(13); Chr\$(10);

Close #3

End Sub

Public Sub LogDecrypt(filename As String)

Open CryptoLog For Append As #3

Print #3, "Decrypt: "; CurrentUserName; " - Console"; " - "; Trim(filename); " - "; Date; Time(); Chr\$(13); Chr\$(10);

Close #3

End Sub

validate - 1

Option Explicit

Dim YNStyle

Dim killit As Long

Function FileTypeWarning(filename As String) As Boolean

{

Dim Fattr As Long

Dim FileExt As String

Dim VerifyIt As Integer

YNStyle = vbYesNo + vbCritical + vbDefaultButton2

FileExt = UCase(Right(filename, 3))

Select Case FileExt

Case "COM"

VerifyIt = MsgBox(filename + Chr\$(13) + Chr\$(10) + "Encrypting this file may cause problems with one or more applications." + Chr\$(13) + Chr\$(10) + " Are you sure you want to continue?", YNStyle)

If VerifyIt = 6 Then

FileTypeWarning = True

Else

FileTypeWarning = False

End If

Case "DAT"

VerifyIt = MsgBox(filename + Chr\$(13) + Chr\$(10) + "Encrypting this file may cause problems with one or more applications." + Chr\$(13) + Chr\$(10) + " Are you sure you want to continue?", YNStyle)

If VerifyIt = 6 Then

FileTypeWarning = True

Else

FileTypeWarning = False

End If

Case "EXE"

VerifyIt = MsgBox(filename + Chr\$(13) + Chr\$(10) + "Encrypting this file may cause problems with one or more applications." + Chr\$(13) + Chr\$(10) + " Are you sure you want to continue?", YNStyle)

If VerifyIt = 6 Then

FileTypeWarning = True

Else

FileTypeWarning = False

End If

Case "INF"

VerifyIt = MsgBox(filename + Chr\$(13) + Chr\$(10) + "Encrypting this file may cause problems with one or more applications." + Chr\$(13) + Chr\$(10) + " Are you sure you want to continue?", YNStyle)

If VerifyIt = 6 Then

FileTypeWarning = True

Else

FileTypeWarning = False

End If

Case "INI"

VerifyIt = MsgBox(filename + Chr\$(13) + Chr\$(10) + "Encrypting this file may cause problems with one or more applications." + Chr\$(13) + Chr\$(10) + " Are you sure you want to continue?", YNStyle)

If VerifyIt = 6 Then

FileTypeWarning = True

Else

FileTypeWarning = False

End If

validate - 2

Case "INF"

```
VerifyIt = MsgBox(filename + Chr$(13) + Chr$(10) + "Encrypting this file may cause problems with one or more applications." + Chr$(13) + Chr$(10) + " Are you sure you want to continue?", YNStyle)
If VerifyIt = 6 Then
    FileTypeWarning = True
Else
    FileTypeWarning = False
End If
```

Case "OCX"

```
VerifyIt = MsgBox(filename + Chr$(13) + Chr$(10) + "Encrypting this file may cause problems with one or more applications." + Chr$(13) + Chr$(10) + " Are you sure you want to continue?", YNStyle)
If VerifyIt = 6 Then
    FileTypeWarning = True
Else
    FileTypeWarning = False
End If
```

Case "SYS"

```
VerifyIt = MsgBox(filename + Chr$(13) + Chr$(10) + "Encrypting this file may cause problems with one or more applications." + Chr$(13) + Chr$(10) + " Are you sure you want to continue?", YNStyle)
If VerifyIt = 6 Then
    FileTypeWarning = True
Else
    FileTypeWarning = False
End If
```

Case "VXD"

```
VerifyIt = MsgBox(filename + Chr$(13) + Chr$(10) + "Encrypting this file may cause problems with one or more applications." + Chr$(13) + Chr$(10) + " Are you sure you want to continue?", YNStyle)
If VerifyIt = 6 Then
    FileTypeWarning = True
Else
    FileTypeWarning = False
End If
```

Case "VBX"

```
VerifyIt = MsgBox(filename + Chr$(13) + Chr$(10) + "Encrypting this file may cause problems with one or more applications." + Chr$(13) + Chr$(10) + " Are you sure you want to continue?", YNStyle)
If VerifyIt = 6 Then
    FileTypeWarning = True
Else
    FileTypeWarning = False
End If
```

Case "DLL"

```
killit = MsgBox("This is a Program Library, Not a User File, and Will Not Be Encrypted", vbCritical + vbOKOnly, "Inteligard Quick Encrypt")
FileTypeWarning = False
```

Case "LNK"

```
killit = MsgBox("This is a Short Cut Link, Not a File. Please Locate the File and Re-try", vbCritical + vbOKOnly, "Inteligard Quick Encrypt")
FileTypeWarning = False
```

Case Else

```
FileTypeWarning = True
```

End Select

validate - 3

End Function

Public Sub WarningMsgBox()

Dim VerifyIt

(

End Sub

frmMain - 1

Marz Carisch

Option Explicit

Private mvarPathToFiles As String

Private Sub cmdDestroy_Click()

MsgBox "Warning! All files will be destroyed, permanently. There is no way to recover destroyed files."

End Sub

Private Sub cmdSelect_Click()

Dim szFile As String

' Handle dismissal of the file dialog via cancel

On Error GoTo msLame:

With cd

.Filter = "All Files (*.*)|*.*"

.Flags = .Flags Or cd1OFNHideReadOnly Or cd1OFNAllowMultiselect Or cd1OFNExplorer

.ShowOpen

If Len(.FileName) = 0 Then

Exit Sub

End If

szFile = .FileName

AddFiles szFile

End With

msLame:

End Sub

Private Sub AddFiles(ByVal szBuffer As String)

' Parse out the file names in the buffer string and add them to the list box.

Dim p As Long

Dim a As String

Dim szPath As String, szFile As String

If szBuffer <> "" Then

lstFiles.Clear

p = InStr(1, szBuffer, Chr\$(0), vbBinaryCompare)

' Parse out all the names in the buffer

If p Then

' Multiple files - extract the path first.

mvarPathToFiles = Left\$(szBuffer, p - 1)

szBuffer = Right\$(szBuffer, Len(szBuffer) - p)

' Now loop out the rest of the files

Do

p = InStr(1, szBuffer, Chr\$(0), vbBinaryCompare)

If p Then

a = Left\$(szBuffer, p - 1)

Else

a = szBuffer

End If

' Remove the current filename

szBuffer = Right\$(szBuffer, Len(szBuffer) - p)

' See if we got something to work with.

If a <> "" Then

lstFiles.AddItem a

End If

Loop While p

Else

' Single file - separate the path and file name

mvarPathToFiles = GetFilePath(szBuffer)

' Get the file name

szFile = GetFileName(szBuffer)

' Add the file to the list box

lstFiles.AddItem szFile

End If

End If

txtPath = mvarPathToFiles

End Sub

Public Function GetFileExtension(ByVal szFile As String) As String

' Returns just the extension from a file

Dim X As Integer

Dim a As String

' Parse backward for first period in string

For X = Len(szFile) To 1 Step -1

If Mid\$(szFile, X, 1) = "." Then Exit For

Next

' If a period was found, use it.

If X Then

GetFileExtension = Right\$(szFile, Len(szFile) - X)

Else

GetFileExtension = ""

End If

End Function

Private Function GetFileName(ByVal szName As String) As String

' Returns the file name without the path

Dim pStart As Long

Dim pEnd As Long

Dim a As String

Dim szSearchChar As String

szSearchChar = "\"

' Find the first path separator in the path

pStart = InStr(1, szName, szSearchChar)

If pStart = 0 Then

' No slashes found. Return the whole thing.

GetFileName = szName

Exit Function

End If

' If one was found, loop

Do While pStart > 0

pStart = InStr(1, szName, szSearchChar)

If pStart > 0 Then

' There was a sep char. Save all chars to the right of it.

szName = Right\$(szName, Len(szName) - pStart)

End If

' Loop for another path char.

pStart = InStr(1, szName, szSearchChar)

Loop

GetFileName = szName

End Function

Private Function GetFilePath(ByVal szFile As String)

' Returns just the path

Dim szTemp As String

szTemp = GetFileName(szFile)

GetFilePath = Left\$(szFile, Len(szFile) - Len(szTemp))

End Function

Private Sub lstFiles_OnClick()

lstFiles.RemoveItem lstFiles.ListIndex

If lstFiles.ListCount = 0 Then

EnableButtons = False

frmMain - 3

```
Else
    EnableButtons True
End If
```

End Sub

```
Private Function EnableButtons(bState As Boolean)
    ' Enable or disable the command buttons as required.
    cmdEncrypt.Enabled = bState
    cmdDecrypt.Enabled = bState
    cmdDestroy.Enabled = bState
End Function
```

Location

lstFiles

Select Files...

Encrypt

Decrypt

Destroy

frmLogin - 1

Option Explicit

' Login security access form.
' Failure to login correctly, or canceling this form will end the program.
' The only way to get to the main form is to enter the correct master password.
' The master password should be stored as a SHA'd value in the encrypted master keyfile.
' . activity can be logged to a log file.

' Maximum number of tries the user is permitted before logout.
Const MAX_TRIES = "E"

' Title of all message boxes displayed by this form.
Const MSGBOX_TITLE = "Login"

Public Caller As CPermission
Public WithEvents ReaderEvent As CReaderEngine
' Global log object
Dim AccessLog As New CErrorLog

Private Sub Form_GotFocus()
' Put us in front of everyone.
Dim t As New CWindowServices
t.SetForeground Me.hWnd
t.SetWinOnTop Me.hWnd, True

Me.txtPassword.SetFocus
End Sub

Private Sub Form_Load()
Dim szBuffer As String
Dim lSize As Long

' Initialize a buffer for API call return.
szBuffer = Space\$(255)
lSize = Len(szBuffer)

' Get the current user name off the card
txtUserName.Text = GetUserName

' Check if logging is enabled.
'SetLogOperation AccessLog
' Log the current username logged in before it gets changed.
AccessLog.Log "Card Inserted - initial user: " & txtUserName
' Put us in front of everyone.
Dim t As New CWindowServices
t.SetForeground Me.hWnd

End Sub

Private Sub Form_Unload(Cancel As Integer)
' Cleanup
Set ReaderEvent = Nothing
Set Caller = Nothing
End Sub

Private Sub cmdCancel_Click()
' No login, no access.

AccessLog.Log "Login Canceled - User: " & txtUserName
Caller.Permission = False
' Evil global boolean flag Used in WaitForServerInit.
' This gets us out of the timeout loop if the user
' explicitly cancels the login.
'gLoginCanceled = True
Inload Me

End Sub

Private Sub cmdOK_Click()

Static TryCount As Long

```

If Not Validate(txtPassword) Then
    ' The first attempt was not valid. Count the login failure.
    TryCount = TryCount + 1
    ' Log the login failure
    AccessLog.Log "Login attempt " & TryCount & " failed. " & "User: " & txtUserName

    ' Obscure the try counter from prying eyes.
    If TryCount >= (Asc(MAX_TRIES) - 64) Then
        ' Log the violation to the event log.
        AccessLog.Log "Trycount exceeded. Keyfile shredded. " & "User: " & txtUserName
        ' Place attack prevention code here. IE shred keyfile, etc.

        MsgBox "Maximum log in attempts exceeded. Access denied. ", , MSGBOX_TITLE
        ' Quit the program, or lock the system here.
        Caller.Permission = False
        AccessLog.Log "Maximum log in attempts exceeded. Access denied. "
        Unload Me

    Else
        ' Maximum count not reached yet. Allow another try.
        MsgBox "Invalid Password, try again.", , MSGBOX_TITLE
        ' Set the focus back on the password box and select the old password for deletion.
        txtPassword.SetFocus
        txtPassword.SelStart = 0
        txtPassword.SelLength = Len(txtPassword.Text)

    End If
Else
    ' User password validated okay. Allow access.
    ' Log the successful log in of user
    AccessLog.Log "Login attempt " & TryCount & " Succeeded." & "User: " & txtUserName
    Caller.Permission = True

    Unload Me
End If

```

End Sub

```

Private Function Validate(szPassword As String) As Boolean
    ' Validate the user's password against the sha value
    Dim szMasterPassword As String
    Dim szKeyFile As String

```

```

    ' Load the password from the card
    szMasterPassword = GetPassword

    ' Clean up the user's password.
    szPassword = Trim(szPassword)

    If szPassword = szMasterPassword Then
        Validate = True
    Else
        Validate = False
    End If

    ' Don't allow blank passwords
    If szPassword = "" Then Validate = False

```

End Function

```

Private Sub ReaderEvent_CardRemoved()
    ' Sorry Charlie.
    cmdCancel_Click
End Sub

```



IntelliGard

User Name:

Password:




EXHIBIT Z-11


IntelliGard Smart Card

Administration

Declaration of Stephen Zizzi



Please insert a
Master Key card.



lstOtherKeys

Master Key Card

User Key Card

User Name:

Password:

User Access:

cmbUserType

Update Card

☐ Hardware

☐ SoftCard

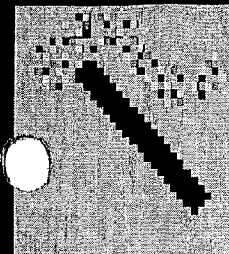
☐ Batch Keys

Smart Card

Admin

3/97





Welcome to the Master Key duplication wizard.
This wizard will help you create an archive
copy of your master key, which can be kept
off site.

Insert the Master Key card which you wish
to duplicate, and hit "Next >"

Please note that once the duplicate card has been created, you
will not be able to add new keys to it, even if the original
Master Key card had open key slots when the duplicate was



frmMain - 1

' Smart card administrator - GemPlus and Card Logix cards
' Steve Zizzi March 1997

Option Explicit

Private mvarMasterCard As CSmartCard
Private mvarOtherCard As New CSmartCard
Private mvarBatchCard As New CSmartCard
Private WithEvents ReaderEvent As CReaderEngine

Private mvarMasterPresent As Boolean ' Flags that the master card is present.

Private Sub Form_Load()

Dim nRet As Long
' Pick up the reader events
Set ReaderEvent = mvarReader
' No menu items available
Me.mnuCard.Enabled = False
Me.mnuMaster.Enabled = False

' Start with burn button disabled.
cmdCreateCard.Enabled = False
' And hide the master keys window
frameNoKeys.Visible = True
' Set up the display
imgSmartCard(1) = Me.imgSmartCard(0)

' Set the user type combo box
cmbUserType.AddItem "User"
cmbUserType.AddItem "System Administrator"

' Do we need to show the SoftCard interface?
If mvarReader.SC Then
optDevice(1).Value = True
frmCardSelect.Show
Else
' Set the device switch to indicate the current device.
optDevice(0).Value = True
End If

End Sub

Private Sub Form_Unload(Cancel As Integer)

' Cleanup
mvarReader.CloseReader
Unload frmCardSelect
'Unload frmDebug
Set mvarMasterCard = Nothing
Set mvarOtherCard = Nothing
Set mvarBatchCard = Nothing
Set ReaderEvent = Nothing
Set mvarReader = Nothing
Set mvarSmartCard = Nothing
End

End Sub

Private Sub cmdCreateCard_Click()

' Commit the data to the card being created.

Dim Header As New CCardHeader
Dim Card As New CSmartCard

Dim nRet As Long
Dim a As String
Dim szType As String

If Not ValidateForm Then

```

' Set up the card header data
a = GetMasterIDString
' Make sure this is not a master card
If Left$(a, 1) = "M" Then
    MsgBox "Please insert a user card."
    Exit Sub
End If
' Set the user type
If cmbUserType.Text = cmbUserType.List(1) Then
    szType = TYPE_SYSADMIN
Else
    szType = "U"
End If
Mid$(a, 1, 1) = szType

' Set the locking byte. In this case, don't lock.
Mid$(a, 2, 1) = Chr$(0)
' Set the count of keys on this card.
Mid$(a, 3, 1) = Chr$(mvarOtherCard.keys.Count)

Header.MasterIDString = a          ' Set the master key string
If mvarOtherCard.Fields("Counter").Data = "" Then
    Header.TryCount = Chr$(0)      ' Initialize the fail counter.
End If

Header.UserName = txtUserName      ' Assign the user name
Header.UserPassword = txtPassword  ' Assign the password
Header.Reserved = Chr$(0)          ' Force this field to be reserved
If mvarOtherCard.Fields("IntelligardID").Data = "" Then
    Header.IntelligardID = GetNewID ' Get the serial number
End If

' Write the data into the card structure
nRet = WriteHeader(Card, Header)
nRet = WriteKeys(nRet, Card, mvarOtherCard.keys)
frmServer.Show
' Tell the card to write itself.
Card.WriteCard frmServer.pbl
' Reality. What the hell is it?
Unload frmServer
Me.sbStatus.Panels("CardStatus") = "Keys have been written to card."
End If

```

End Sub

```

Private Function ValidateForm() As Boolean
' Check the data on the form before writing the card.
Dim bFail As Boolean

If Me.txtPassword = "" Then bFail = True
If Me.txtUserName = "" Then bFail = True
If lstOtherKeys.ListCount = 0 Then bFail = True

ValidateForm = bFail

```

End Function

```

Private Function LoadMasterCard() As Boolean
' An existing master card has been inserted, and
' succesfully booted. Check if it's new.
Dim nRet As Long
Dim field As New CField

If IsNewMaster And Not IsMasterLocked Then
    ' Allow initialization of the master card.
    ShowWizard frmMasterWizard
    LoadMasterCard = True
Else
    ' Ask for the password to the card.

```

```

Dim Please As New CPermission
Set frmLogin.Caller = Please
' Pass a ref to the reader
Set frmLogin.ReaderEvent = mvarReader
frmLogin.Show vbModal
' See if the user entered the correct password
If Please.Permission Then
    ' This is not a new card. Attempt to load the keys
    sbStatus.Panels("CardStatus") = "Loading keys..."

    If LoadMasterKeys = 0 Then
        sbStatus.Panels("CardStatus") = "Master Keys have been loaded."
        frameNoKeys.Visible = False
        Set field = mvarMasterCard.Fields("UserName")
        sbStatus.Panels("CurrentMaster") = field.Data
        sbStatus.Panels("CurrentAccess") = "Security Administrator"
        mnuMaster.Enabled = True
        mnuCard.Enabled = False
        mvarMasterPresent = True
        Set mvarBatchCard = Nothing
        chkBatching.Value = vbUnchecked
        LoadMasterCard = 0
    Else
        ' An error occurred. Master keys could not be loaded.
        sbStatus.Panels("CardStatus") = "Master keys could not be loaded." & " Error # " &
"0x" & Hex$(nRet)
        LoadMasterCard = 1
    End If

Else
    ' Don't let the user use the software.
    sbStatus.Panels("CardStatus") = "Access denied."
End
End If
End If

```

End Function

Private Function LoadOtherCard()

' An existing user card has been inserted, and successfully booted.
' Don't care what it looks like. Just try to load it.

```

Dim nRet As Long
Dim szType As String
Dim szStatus As String
Dim field As New CField

```

' Only allow the user card to be loaded if a master is present.

If mvarMasterPresent Then

```

' This is not a new card. Attempt to load the keys
sbStatus.Panels("CardStatus") = "Loading keys..."

```

If LoadOtherKeys = 0 Then

```

Set field = mvarOtherCard.Fields("UserName")
txtUserName = field.Data
szType = GetCardType
' Display the card type (if any) in the combo box.
If szType = TYPE_SYSADMIN Then
    cmbUserType.Text = cmbUserType.List(1)
Else
    cmbUserType.Text = cmbUserType.List(0)
End If

```

```

If field.Data = "" Then
    szStatus = "Smart card is ready"
Else

```

```

    szStatus = "Existing keys have been loaded."
End If

```

```

' If there is no password on the card yet, one will need to be set.
Set field = mvarOtherCard.Fields("UserPassword")

```

```

If field.Data = "" Then
    txtPassword.PasswordChar = ""
    txtPassword.Enabled = True
    mnuCChangePassword.Enabled = False

```

```

Else
    txtPassword.PasswordChar = "*"
    txtPassword.Enabled = False
    mnuCChangePassword.Enabled = True

```

```

End If
txtPassword.Text = field.Data

```

```

sbStatus.Panels("CardStatus") = szStatus
mnuCard.Enabled = True
mnuMaster.Enabled = False

```

```

' Handle batch card creation
If chkBatching.Value = vbChecked Then
    AddBatchKeys
Else
    Set mvarBatchCard = Nothing
End If

```

```

LoadOtherCard = 0

```

```

Else
    ' An error occurred. Master keys could not be loaded.
    sbStatus.Panels("CardStatus") = "Keys could not be loaded." & " Error # " & "0x" & Hex
$(nRet)

```

```

LoadOtherCard = 1

```

```

End If

```

```

End If

```

```

End Function

```

```

Private Function LoadOtherKeys() As Long

```

```

' Read the keys off a user card

```

```

    Dim key As CNewKey

```

```

    ' Instantiate a smart card data struct

```

```

    Set mvarOtherCard = New CSmartCard

```

```

    ' Load it.

```

```

    frmServer.Show

```

```

    LoadOtherKeys = mvarOtherCard.ReadCard(frmServer.pb1)

```

```

    Unload frmServer

```

```

End Function

```

```

Private Function AddBatchKeys() As Boolean

```

```

' Add keys from the batch set to the current card

```

```

    Dim mKey As New CNewKey

```

```

    Dim oKey As New CNewKey

```

```

    Dim bFound As Boolean

```

```

    ' Create the batch key structure

```

```

    If mvarBatchCard Is Nothing Then

```

```

        Set mvarBatchCard = New CSmartCard

```

```

    Else

```

```

        ' Search the batch collection for keys that aren't in the new card.

```

```

        For Each mKey In mvarBatchCard.keys

```

```

            ' Search the user keys for an existing key by key value

```

```

            For Each oKey In mvarOtherCard.keys

```

```

                If oKey.KeyValue = mKey.KeyValue Then

```

```

                    bFound = True

```

```

                    Exit For

```

```

                End If

```

```

            Next

```

```

            ' If the key in the batch collection doesn't exist on this card, add it.

```

```

            If Not bFound Then

```

```

                mvarOtherCard.keys.Add mKey, mKey.KeyName

```

```

        ' Update the display
        DisplayKeys lstOtherKeys, mvarOtherCard.keys

```

```

    End If

```

```

Next

```

```

sbStatus.Panels("CardStatus") = "Batch keys have been added"

```

```

End If

```

```

End Function

```

```

Private Function LoadMasterKeys() As Long

```

```

    ' Read the keys off the master card

```

```

    ' Instantiate a smart card data struct

```

```

    Set mvarMasterCard = New CSmartCard

```

```

    ' Load it.

```

```

    frmServer.Show

```

```

    LoadMasterKeys = mvarMasterCard.ReadCard(frmServer.pb1)

```

```

    Unload frmServer

```

```

End Function

```

```

Private Sub lstMasterKeys_DblClick()

```

```

    Dim mKey As New CNewKey

```

```

    Dim oKey As New CNewKey

```

```

    Dim bFound As Boolean

```

```

    Set mKey = mvarMasterCard.keys(lstMasterKeys.List(lstMasterKeys.ListIndex))

```

```

    ' Search the user keys for an existing key by key value

```

```

    For Each oKey In mvarOtherCard.keys

```

```

        If oKey.KeyValue = mKey.KeyValue Then

```

```

            bFound = True

```

```

            Exit For

```

```

        End If

```

```

    Next

```

```

    If Not bFound Then

```

```

        mvarOtherCard.keys.Add mKey, mKey.KeyName

```

```

        ' Maintain a batch card which has only the new master keys.

```

```

        mvarBatchCard.keys.Add mKey, mKey.KeyName

```

```

        ' Update the display

```

```

        DisplayKeys lstOtherKeys, mvarOtherCard.keys

```

```

    Else

```

```

        MsgBox "This key already exists on the user card."

```

```

    End If

```

```

    ' If keys exist on the card, enable the update button.

```

```

    If lstOtherKeys.ListCount > 0 Then

```

```

        Me.cmdCreateCard.Enabled = True

```

```

    Else

```

```

        Me.cmdCreateCard.Enabled = False

```

```

    End If

```

```

End Sub

```

```

Private Sub lstOtherKeys_DblClick()

```

```

    Dim key As New CNewKey

```

```

    Dim bFound As Boolean

```

```

    ' Make sure this key exists on the master before deletion. Don't want

```

```

    ' to delete a unique key.

```

```

    For Each key In mvarMasterCard.keys

```

```

        If key.KeyName = lstOtherKeys.List(lstOtherKeys.ListIndex) Then

```

```

            bFound = True

```

```

            Exit For

```

```

        End If

```

```

    Next

```

```

    If bFound Then

```

```

        mvarOtherCard.keys.Remove lstOtherKeys.List(lstOtherKeys.ListIndex)

```

```

' Blow off errors due to key not existing in batch collection
On Error Resume Next
    mvarBatchCard.keys.Remove lstOtherKeys.List(lstOtherKeys.ListIndex)
On Error GoTo 0
' Update the display
DisplayKeys lstOtherKeys, mvarOtherCard.keys
Else
' Don't allow a unique key (i.e. one not found on the master card) to
' be deleted. In theory, this should happen. But, we all know users.
MsgBox "The selected key cannot be deleted. It is a unique key."
End If

' If keys exist on the card, enable the update button.
If lstOtherKeys.ListCount > 0 Then
    Me.cmdCreateCard.Enabled = True
Else
    Me.cmdCreateCard.Enabled = False
End If

```

End Sub

Private Sub mnuCChangePassword_Click()

```

' Change an existing user password
' This menu item will only be available if a current user card is in
' the reader, and has a password that can be changed.

Me.txtPassword.Enabled = True      ' Allow editing
Me.txtPassword = ""                ' Clear the existing password
Me.txtPassword.PasswordChar = ""   ' Set it to clear text
Me.txtPassword.SetFocus            ' Put the cursor in the text box.

```

End Sub

Private Sub mnuFExit_Click()

```

Unload Me
End Sub

```

Private Sub mnuMAddKeys_Click()

```

ShowWizard frmMasterEdit

```

End Sub

Private Sub mnuMDuplicate_Click()

```

ShowWizard frmMasterDuplicate

```

End Sub

Private Sub ShowWizard(frmWizard As Form)

```

' Show the wizard form passed in

' Don't watch the reader while the wizard is showing
'Set ReaderEvent = Nothing
' Simulate modality
frmMain.Enabled = False
frmWizard.Show , Me
' Pick up the reader events again
'Set ReaderEvent = mvarReader

```

End Sub

Private Sub optDevice_Click(index As Integer)

```

Dim nRet As Long
' Abort reacting to reader events during swap process
Set ReaderEvent = Nothing

' Force the engine to set its interface type to what it will be

```



```

' On next instantiation.
If index = 1 Then
    ' Engine will restart with the software interface.
    mvarReader.SC = True
    frmCardSelect.Show
Else
    ' Engine will restart with the hardware interface.
    mvarReader.SC = False
    Unload frmCardSelect
End If

' Close the reader interface
CloseReader
Set mvarReader = Nothing

' Create a new card reader engine
Set mvarReader = CreateObject("ReaderEngine.CReaderEngine")
' Pick up the new reader events
Set ReaderEvent = mvarReader

' Turn it on.
nRet = OpenReader
If nRet <> 0 Then
    ' Error occurred during initialization of the reader
    MsgBox "Card reader failed to initialize." & "Error # " & "0x" & Hex$(nRet)
    ' No point in continuing, can't talk to the card.
    ' End
End If

```

End Sub

Private Sub ReaderEvent_CardInserted()

```

Dim nRet As Long
Dim szStatus As String

' Attempt to boot the card
sbStatus.Panels("CardStatus") = "Bootting card..."
' Software is booted in the frmCardSelect imgCard_Click event.
If Not mvarReader.SC Then AttemptBoot
nRet = InitializeCard
' Check the results
If nRet Then
    ' Card failed to boot.
    szStatus = "Card failed to boot." & " Error # " & "0x" & Hex$(nRet)
    Exit Sub
Else
    szStatus = "Smart Card Ready"
End If

' Update the status bar
sbStatus.Panels("CardStatus") = szStatus

' Determine what kind of card has been inserted. Is it a master card?
If IsMasterCard Then
    ' Attempt to load it.
    If Not LoadMasterCard Then
        ' Display the keys just loaded.
        DisplayKeys lstMasterKeys, mvarMasterCard.keys
    End If
Else
    ' See if they have permission to edit a user card.
    If mvarMasterPresent Then
        ' It's some other type of card.
        If Not LoadOtherCard Then
            ' Display the keys just loaded, if any.
            DisplayKeys lstOtherKeys, mvarOtherCard.keys
        End If
    End If

```

' If keys exist on the card, enable the update button.

If lstOtherKeys.ListCount > 0 Then

Me.cmdCreateCard.Enabled = True

Else

Me.cmdCreateCard.Enabled = False

End If

Else

MsgBox "You must sign on with a Master Key card to use this program."

End If

End If

End Sub

Private Sub ReaderEvent_CardRemoved()

' Update the status bar

sbStatus.Panels("CardStatus") = "No card detected."

' Disable the master and card menus, as there's no card to work with.

mnuMaster.Enabled = False

mnuCard.Enabled = False

cmdCreateCard.Enabled = False

End Sub

frmCardSelect - 1

Option Explicit

Private mvarCardState As Boolean

Private Sub cmdNewUser_Click()

CreateNewFile "Blank User"

End Sub

Private Sub cmdNewMaster_Click()

CreateNewFile "Blank Master"

End Sub

Private Sub CreateNewFile(szTemplateName As String)

' Get the name of the file to save

CD.Flags = CD.Flags Or cdIOFNOverwritePrompt Or cdIOFNHideReadOnly

CD.FileName = GetFilePath(CD.FileName) & szTemplateName & ".crd"

' Set CancelError is True

CD.CancelError = True

On Error GoTo ErrHandler

CD.ShowSave

If CD.FileName <> "" Then

FileCopy App.Path & "\" & szTemplateName & ".tpl", CD.FileName

End If

lblKeyFile.Caption = CD.FileTitle

ErrHandler:

'User pressed the Cancel button

Exit Sub

End Sub

Private Sub Form_Load()

' Set the card not in the reader

mvarCardState = False

SetCardImage

End Sub

Private Sub cmdLoadCard_Click()

' Set CancelError is True

CD.CancelError = True

On Error GoTo ErrHandler

' Get the name of the data file to read in.

CD.ShowOpen

lblKeyFile.Caption = CD.FileTitle

ErrHandler:

End Sub

Private Sub imgCardState_Click()

'Toggle the card in or out of the reader

mvarCardState = Not mvarCardState

SetCardImage

SetCard

End Sub

Private Sub SetCardImage()

' Set the state of the card image

If mvarCardState Then

imgCardState = imgCardIn

Me.cmdLoadCard.Enabled = False

Me.cmdNewUser.Enabled = False

```
Me.cmdNewMaster.Enabled = False
Else
    imgCardState = imgCardOut
    Me.cmdLoadCard.Enabled = True
    Me.cmdNewUser.Enabled = True
    Me.cmdNewMaster.Enabled = True
End If
```

```
End Sub
```

```
Private Sub SetCard()
```

```
' Sets the card file in or out of the reader.
```

```
If mvarCardState Then
```

```
    If CD.FileName <> "" Then
```

```
        ' Pass in a card data file to boot with.
```

```
        If CD.FileName <> "" Then
```

```
            AttemptBoot CD.FileName
```

```
        End If
```

```
    Else
```

```
        mvarCardState = False
```

```
        SetCardImage
```

```
        MsgBox "Must have keyfile to load."
```

```
    End If
```

```
Else
```

```
' This will simulate pulling the card but leaving the reader open.
```

```
mvarReader.CloseReader
```

```
mvarReader.OpenReader
```

```
End If
```

```
End Sub
```

frmGenerate - 1

Option Explicit

Const ROWS = 6
Const COLUMNS = 6
Const SPACER = 0
Const REQUIRED_DIGITS = 400
Public frmCaller As Form

Private Sub cmdCancel_Click()
 Unload Me
End Sub

Private Sub cmdStart_Click()
 Dim bFlag As Boolean

 ' Don't allow leading or trailing spaces in a key name.
 txtKeyName = Trim\$(txtKeyName)

 If txtKeyName = "" Then
 MsgBox "Please provide a name for the key to be created."
 bFlag = True
 End If

 ' Validate the key name that the user has supplied.
 If Not bFlag Then
 If ValidateKeyName(txtKeyName) Then
 MsgBox "A key with this name already exists. Key names must be unique."
 bFlag = True
 End If
 End If

 If Not bFlag Then
 ' Make sure the matrix is black.
 BlackenMatrix
 ' Enable the color matrix
 EnableMatrix True
 ' Restart the progress bar
 pbProgress.Value = 0
 cmdStart.Enabled = False
 ' Don't allow the cancel button to be hit by the space bar.
 cmdCancel.Cancel = False
 cmdCancel.Default = False

 ' Disable any key name editing.
 txtKeyName.Enabled = False
 End If

End Sub

Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
 ' Re-map the space bar to avoid having the cancel key pressed.
 If (KeyCode = 32) And (cmdStart.Enabled = False) Then KeyCode = 65

End Sub

Private Sub Form_KeyPress(KeyAscii As Integer)

 Dim R As Long, G As Long, B As Long
 Dim nBoxes As Long
 Dim nIndex As Long

 ' Make sure the Start button is pressed.

 If cmdStart.Enabled = False Then

 ' Pick random values for RGB using the system timer and the key pressed as seeds

 Randomize: R = (Int((256 * Rnd) + 1) - 1) Xor CByte(Abs(KeyAscii * (Int((256 * Rnd) + 1) - 1)) Mod 256)

 Randomize: G = (Int((256 * Rnd) + 1) - 1) Xor CByte(Abs(KeyAscii * (Int((256 * Rnd) + 1) - 1)) Mod 256)

 Randomize: B = (Int((256 * Rnd) + 1) - 1) Xor CByte(Abs(KeyAscii * (Int((256 * Rnd) + 1) - 1)) Mod 256)

```

' Calculate how many boxes are on the screen, adjust for one off.
nBoxes = ROWS * COLUMNS + 1
' Map the random number to a randomly selected box. Seed with timer.
Randomize: nIndex = (Int((nBoxes) * Rnd) + 1) - 1
lblBox(nIndex).BackColor = RGB(R, G, B)
' increment the progress bar
pbProgress.Value = pbProgress.Value + 1
' Check if we're done yet.
If pbProgress.Value = REQUIRED_DIGITS Then
    ' Done. Shut off the matrix to prevent recursion.
    EnableMatrix False
    ' Generate the key
    GenKey
    MsgBox "The key has been generated."
    Unload Me
End If
End If

```

End Sub

Private Sub Form_Load()

```

Dim x As Long
Dim y As Long
Dim nTop As Long
Dim nLeft As Long
Dim n As Long

pbProgress.Value = 0
' Set the progress bar
pbProgress.Max = REQUIRED_DIGITS

' Set the top & left
nTop = lblBox(0).Top: nLeft = lblBox(0).Left

' Load the color boxes and initialize them to black.
For x = 1 To ROWS
    For y = 1 To COLUMNS
        n = ((x - 1) * COLUMNS) + y
        Load lblBox(n)
        lblBox(n).Left = nLeft: lblBox(n).Top = nTop
        lblBox(n).BackColor = 0
        lblBox(n).Visible = True
        nLeft = nLeft + lblBox(0).Width + SPACER
    Next
    ' Set the new top and left offsets for the boxes
    nTop = nTop + lblBox(0).Height + SPACER: nLeft = lblBox(0).Left
Next
' Make sure the matrix is disabled.
EnableMatrix False

```

End Sub

Private Sub lblBox_MouseMove(index As Integer, Button As Integer, Shift As Integer, x As Single, y As Single)

```

' Change the backcolor of the box to a random rgb value
Dim R As Long
Dim G As Long
Dim B As Long

' Generate random value between 0 and 255 for each color.
Randomize: R = (Int((256 * Rnd) + 1) - 1) Xor CByte(Abs(x) Mod 256)
Randomize: G = (Int((256 * Rnd) + 1) - 1) Xor CByte(Abs(y) Mod 256)
Randomize: B = (Int((256 * Rnd) + 1) - 1) Xor CByte(Abs(x) Mod 256 Xor Abs(y) Mod 256)
' Apply the color to the background of the box.
lblBox(index).BackColor = RGB(R, G, B)
' increment the progress bar
pbProgress.Value = pbProgress.Value + 1

```

```

' Check if we're done yet.
If pbProgress.Value = REQUIRED DIGITS Then
    ' Done. Shut off the matrix to prevent recursion.
    EnableMatrix False
    ' Make sure that more than 50 percent of the squares have been changed.
    If Not ValidateSquare Then
        GenKey
        MsgBox "The key has been generated."
        Unload Me
    Else
        ' The user didn't hit enough of the color squares to be useful.
        MsgBox "You must move the mouse over at least fifty percent of the color squares. Please try again."
        cmdStart.Enabled = True
    End If
End If

End Sub

Private Sub EnableMatrix(State As Boolean)
    ' Enable or disable the matrix
    Dim x As Long
    ' Loop through the box array and enable or disable as required.
    For x = 0 To lblBox().Count - 1
        lblBox(x).Enabled = State
    Next
End Sub

Private Function GenKey()
    ' Generate the key based on the matrix
    Dim key As CNewKey
    Set key = frmCaller.CurrentKey
    ' Generate the string to return, and set it.
    With key
        .KeyValue = GenString
        .KeyName = txtKeyName
    End With
End Function

Private Function GenString() As String
    ' Change the RGB color values of the boxes to ascii chars.
    Dim x As Long
    Dim szKey As String
    Dim c As Long

    Dim R As Integer
    Dim G As Integer
    Dim B As Integer

    For x = 1 To lblBox().Count - 1
        ' Get the background color of the box.
        c = lblBox(x).BackColor
        ' Convert the value into RGB values
        R = Component(c, "R")
        G = Component(c, "G")
        B = Component(c, "B")

        szKey = szKey & Chr$(R) & Chr$(G) & Chr$(B)
    Next

    GenString = szKey
End Function

Private Function Component(Color As Long, szComponent As String) As Integer
    ' Return the requested RGB component of the color passed in.

    Dim R As Long
    Dim G As Long
    Dim B As Long

```

```
' Convert Color to RGB:
```

```
B = Color \ 65536
```

```
G = (Color - (B * 65536)) \ 256
```

```
R = Color - ((B * 65536) + (G * 256))
```

```
' Return the selected componet of the color value
```

```
Select Case UCase$(szComponent)
```

```
Case "R"
```

```
Component = R
```

```
Case "G"
```

```
Component = G
```

```
Case "B"
```

```
Component = B
```

```
Case Else
```

```
Component = 0
```

```
End Select
```

```
End Function
```

```
Private Function ValidateKeyName(szKeyName As String) As Boolean
```

```
' Query the master wizard form if it has this key name in its collection
```

```
ValidateKeyName = frmMasterWizard.ValidateKeyName(szKeyName)
```

```
End Function
```

```
Private Function ValidateSquare() As Boolean
```

```
' Check that at least 50 percent of the squares have been changed.
```

```
Dim x As Long
```

```
Dim c As Long
```

```
For x = 1 To lblBox().Count - 1
```

```
' count the number of black squares.
```

```
If lblBox(x).BackColor = 0 Then
```

```
c = c + 1
```

```
End If
```

```
Next
```

```
' Now check that the total number of black squares is less than 50 percent of the total.
```

```
If c > ((lblBox().Count - 1) / 2) - 1 Then
```

```
ValidateSquare = True
```

```
Else
```

```
ValidateSquare = False
```

```
End If
```

```
End Function
```

```
Private Sub BlackenMatrix()
```

```
' Turn all squares in the matrix black.
```

```
Dim x As Long
```

```
For x = 1 To lblBox().Count - 1
```

```
lblBox(x).BackColor = 0
```

```
Next
```

```
End Sub
```

```
Private Sub txtKeyName_KeyPress(KeyAscii As Integer)
```

```
' Start the color matrix if they press enter.
```

```
If KeyAscii = 13 Then cmdStart_Click
```

```
End Sub
```


frmLogin - 1

Option Explicit

' Login security access form.

' Failure to login correctly, or canceling this form will end the program.

' The only way to get to the main form is to enter the correct master password.

' The master password should be stored as a SHA'd value in the encrypted master keyfile.
' activity can be logged to a log file.

' Maximum number of tries the user is permitted before logout.

Const MAX_TRIES = "5"

' Title of all message boxes displayed by this form.

Const MSGBOX_TITLE = "Login"

Public Caller As CPermission

Public WithEvents ReaderEvent As CReaderEngine

' Global log object

Dim AccessLog As New CErrorLog

Private Sub Form_Load()

Dim szBuffer As String

Dim lSize As Long

' Initialize a buffer for API call return.

szBuffer = Space\$(255)

lSize = Len(szBuffer)

' Get the current user name off the card

txtUserName.Text = GetUserName

' Check if logging is enabled.

'SetLogOperation AccessLog

' Log the current username logged in before it gets changed.

AccessLog.Log "Card Inserted - initial user: " & txtUserName

End Sub

Private Sub Form_Unload(Cancel As Integer)

' Cleanup

Set ReaderEvent = Nothing

Set Caller = Nothing

End Sub

Private Sub cmdCancel_Click()

' No login, no access.

AccessLog.Log "Login Canceled - User: " & txtUserName

Caller.Permission = False

Unload Me

End Sub

Private Sub cmdOK_Click()

Static TryCount As Long

If Not Validate(txtPassword) Then

' The first attempt was not valid. Count the login failure.

TryCount = TryCount + 1

' Log the login failure

AccessLog.Log "Login attempt " & TryCount & " failed. " & "User: " & txtUserName

' Obscure the try counter from prying eyes.

If TryCount >= (Asc(MAX_TRIES) - 64) Then

' Log the violation to the event log.

AccessLog.Log "Trycount exceeded. Keyfile shredded. " & "User: " & txtUserName

' Place attack prevention code here. IE shred keyfile, etc.

MsgBox "Maximum log in attempts exceeded. Access denied. ", , MSGBOX_TITLE

' Quit the program, or lock the system here.

Caller.Permission = False

```
AccessLog.Log "Maximum log in attempts exceeded. Access denied. "  
Unload Me
```

```
Else
```

```
' Maximum count not reached yet. Allow another try.  
MsgBox "Invalid Password, try again.", , MSGBOX_TITLE  
' Set the focus back on the password box and select the old password for deletion.  
txtPassword.SetFocus  
txtPassword.SelStart = 0  
txtPassword.SelLength = Len(txtPassword.Text)
```

```
End If
```

```
Else
```

```
' User password validated okay. Allow access.  
' Log the successful log in of user  
AccessLog.Log "Login attempt " & TryCount & " Succeeded." & "User: " & txtUserName  
Caller.Permission = True
```

```
Unload Me
```

```
End If
```

```
End Sub
```

```
Private Function Validate(szPassword As String) As Boolean
```

```
' Validate the user's password against the sha value
```

```
Dim szMasterPassword As String
```

```
Dim szKeyFile As String
```

```
' Load the password from the card  
szMasterPassword = GetPassword
```

```
' Clean up the user's password.  
szPassword = Trim(szPassword)
```

```
If szPassword = szMasterPassword Then  
    Validate = True
```

```
Else
```

```
    Validate = False
```

```
End If
```

```
' Don't allow blank passwords
```

```
If szPassword = "" Then Validate = False
```

```
End Function
```

```
Private Sub ReaderEvent_CardRemoved()
```

```
' Sorry Charlie.
```

```
Caller.Permission = False
```

```
End Sub
```

frmMasterDuplicate - 1

Option Explicit

Const FRAME_INSERT_MASTER = 0

Const FRAME_MASTER_PASS = 1

Const FRAME_SHOW_MASTER_KEYS = 2

Const FRAME_INSERT_BLANK = 3

Const FRAME_TRANSPORT = 4

Const FRAME_FINISH = 5

Const FRAME_DONE = 6

Const FRAME_BORDER = 0

Const FRAME_OFF = -10000

Const FRAME_LEFT = 60

Const FRAME_TOP = 60

Const REQUIRED_PASSWORD_LEN = 6

Const AVAILABLE_KEY = "...(key slot available)..."

' Display variables

Private FrameNumber As Long

Private LastFrame As Long

' Special members

'Private Keys As New Collection

Private mvarNewKey As New CNewKey

Private WithEvents ReaderEvent As CReaderEngine

Private mvarMasterCard As CSmartCard

Private mvarOkToRemove As Boolean

Private Sub Form_Load()

Dim x As Long

' Set up a ref to the reader object

Set ReaderEvent = mvarReader

' Initialize the back and next keys to the startup state.

cmdBack.Enabled = False

cmdNext.Enabled = True

cmdFinish.Enabled = False

Dim nRet As Long

' Save resources by picking the picture off the main form.

imgSmartCard.Picture = frmMain.imgSmartCard(0).Picture

' Initialize the number of the last frame. The "done" frame
' is not counted, either.

LastFrame = Frame().Count - 2

' Set the first frame to be shown.

FrameNumber = 0

' Make sure all frames are hidden

For x = 0 To Frame().Count - 1

HideFrame x

Next

' Remove the borders

For x = 0 To FRAME_DONE

Frame(x).BorderStyle = FRAME_BORDER

Next

' See if the Next button should be enabled

If FrameNumber = LastFrame Then cmdNext.Enabled = False

' Show the first frame

ShowFrame FrameNumber

End Sub

Private Sub Form_Unload(Cancel As Integer)

' Clean up.

If Not ReaderEvent Is Nothing Then

Set ReaderEvent = Nothing

End If

If Not mvarMasterCard Is Nothing Then

Set mvarMasterCard = Nothing

End If

frmMasterDuplicate - 2

```
' Re-enable main
frmMain.Enabled = True
End Sub
```

```
Public Property Get CurrentKey() As CNewKey
' Turn a reference to this object's current key
Set CurrentKey = mvarNewKey
End Property
```

```
Private Sub cmdCancel_Click()
Unload Me
End Sub
```

```
Private Sub cmdFinish_Click()
' Write the keys to the master card, and close the card.
```

```
Dim nRet As Long
```

```
' One last check before writing the keys.
```

```
nRet = MsgBox("Are you sure you want to create the duplicate Master Key card now?", vbYesNo)
```

```
If nRet = vbYes Then
```

```
frameStatus.Visible = True
```

```
BurnBabyBurn.1
```

```
End If
```

```
End Sub
```

```
Private Sub BurnBabyBurn(nStructID As Long)
```

```
' Dispatch the correct burn function
```

```
Select Case nStructID
```

```
Case 1
```

```
' Dispatch
```

```
BurnStructOne
```

```
Case Else
```

```
End Select
```

```
End Sub
```

```
Private Sub BurnStructOne()
```

```
Dim Header As New CCardHeader
```

```
Dim Card As New CSmartCard
```

```
Dim nRet As Long
```

```
Dim a As String
```

```
Dim field As Cfield
```

```
Dim szUserName As String
```

```
Dim szUserPassword As String
```

```
Set field = mvarMasterCard.Fields("UserName")
```

```
szUserName = field.Data
```

```
Set field = mvarMasterCard.Fields("UserPassword")
```

```
szUserPassword = field.Data
```

```
a = GetMasterIDString
```

```
' Set the number of keys as permanent.
```

```
Mid$(a, 3, 1) = Chr$(mvarMasterCard.keys.Count)
```

```
' Mark it as locked
```

```
Mid$(a, 2, 1) = Chr$(1)
```

```
Header.MasterIDString = a
```

```
' Set the master key string locked.
```

```
Header.TryCount = "0"
```

```
' Clear the fail counter
```

```
Header.UserName = szUserName
```

```
' Assign the user name
```

```
Header.UserPassword = szUserPassword
```

```
' Assign the admin password
```

```
Header.Reserved = Chr$(0)
```

```
Header.IntelligardID = GetNewID
```

```
' Get the serial number
```

```
nRet = WriteHeader(Card, Header)
```

```
nRet = WriteKeys(nRet, Card, mvarMasterCard.keys)
```

```
If Card.WriteCard(pbl) <> 0 Then
```

frmMasterDuplicate - 3

```
' Error in key creation.
MsgBox "There has been an error during master key creation. Please contact your system ac
ministrato."
Else
' Success. Leave them no choice but to remove the smart card.
Me.cmdNext = False
Me.cmdBack.Enabled = False
Me.cmdCancel.Enabled = False
Me.cmdFinish.Enabled = False
ShowFrame FRAME_DONE
End If

End Sub

Private Sub cmdBack_Click()

' Hide the current frame
HideFrame FrameNumber
' Go back to the last frame.
FrameNumber = FrameNumber - 1

' Show the new frame
ShowFrame FrameNumber

If FrameNumber = 0 Then
' At the first frame, so disable the "back" button.
cmdBack.Enabled = False
End If
If FrameNumber < LastFrame Then
' Not at the last frame. Disable the finish button.
cmdFinish.Enabled = False
End If

' If you're going backward, you're coming from an enabled frame, so
' make sure the next button is enabled, in case it was the last frame.
If FrameNumber <> FRAME_TRANSPORT Then
cmdNext.Enabled = True
End If
End Sub

Private Sub cmdNext_Click()

' Check the current status of the data before moving on.
If Not Validate(FrameNumber) Then

' Hide the current frame
HideFrame FrameNumber
' Advance to the next frame
FrameNumber = FrameNumber + 1
' Show the next frame
ShowFrame FrameNumber
' Check if the frame should be disabled.
If FrameNumber = LastFrame Then
' At the last frame, so disable the "next" button, enable finish button
cmdNext.Enabled = False
cmdFinish.Enabled = True
End If
' Check if the back button should be enabled
If FrameNumber > 0 Then
cmdBack.Enabled = True
End If

End If

End Sub

Private Sub HideFrame(n As Long)
' Hide the frame passed in N

Frame(n).Left = FRAME_OFF
```

End Sub

Private Sub ShowFrame(n As Long)

' Hide the frame passed in N

Frame(n).Left = FRAME_LEFT

Frame(n).Top = FRAME_TOP

Frame(n).Width = Frame(0).Width

Frame(n).Height = Frame(0).Height

' Set special case conditions here.

SetFrame n

End Sub

Private Function Validate(FrameNumber As Long) As Boolean

' Apply validation rules to data on frame passed in.

' Returns true if the data is not acceptable.

Select Case FrameNumber

Case FRAME_INSERT_MASTER

' Make sure the user has inserted a master card to duplicate.

If Not IsMasterCard Then

' Tell the user they must insert a master card for duplication.

MsgBox "The card you have inserted is not a Master Key card."

Validate = False

End If

Case FRAME_MASTER_PASS

' Master Password frame. Validate the password.

Validate = ValidatePassword(txtCurrentMasterPass.Text)

Case Else

End Select

End Function

Private Function SetFrame(n As Long)

' Set up special conditions for the frame being shown.

Dim szKeys As String

Dim szSlots As String

Dim t As Long

Select Case n

Case FRAME_INSERT_MASTER

Set mvarMasterCard = Nothing

Case FRAME_MASTER_PASS

' Coming into this frame, forget the current password

ValidatePassword "", True

Me.txtCurrentMasterPass = ""

cmdNext.Enabled = False

Case FRAME_TRANSPORT

cmdNext.Enabled = False

imgLock = imgLocked

txtTransport = ""

txtTransport.SetFocus

ValidatePassword "", True

Case FRAME_SHOW_MASTER_KEYS

Me.Refresh

DoEvents

LoadMasterKeys

Case FRAME_INSERT_BLANK

cmdBack.Enabled = False

' No going back now.

cmdNext.Enabled = False

' Now wait for them to put in a blank master card.

mvarOkToRemove = True

```

Case LastFrame
Case Else
    ' Do nothing

```

```

End Select

```

```

End Function

```

```

Private Function LoadMasterKeys()

```

```

    ' Only load the keys once.
    If mvarMasterCard Is Nothing Then
        ' Instantiate a smart card data struct
        Set mvarMasterCard = New CSmartCard
        ' Load it.
        mvarMasterCard.ReadCard pb
        pb = 0
        Dim key As CNewKey
        ' Clear any existing keys from the box.
        lstKeys.Clear
        ' Load the box with the key names.
        For Each key In mvarMasterCard.keys
            lstKeys.AddItem key.KeyName
        Next
    End If

```

```

End Function

```

```

Private Function ValidatePassword(szPassword As String, Optional Clear As Boolean) As Boolean

```

```

    ' Compare the user's password to the transport password on the card.

```

```

    ' Returns false if the password matches

```

```

    Static szPass

```

```

    ' Forget the password if necessary.

```

```

    If Clear Then szPass = ""

```

```

    If szPass = "" Then

```

```

        szPass = GetPassword

```

```

    End If

```

```

    If szPassword = szPass Then

```

```

        ' Password matches.

```

```

        ValidatePassword = False

```

```

    Else

```

```

        ' Bzzz. Sorry, please drive through.

```

```

        ValidatePassword = True

```

```

    End If

```

```

End Function

```

```

Private Sub ReaderEvent CardInserted()

```

```

    ' The only time they should trigger this event is when putting in a blank

```

```

    ' card on request. Otherwise, removing the card will cancel the wizard.

```

```

    ' Did they put the right card in?

```

```

    If IsNewMaster And Not IsMasterLocked Then

```

```

        ' If they pull this card out, the wizard will cancel.

```

```

        mvarOkToRemove = False

```

```

        cmdNext.Enabled = True

```

```

    Else

```

```

        MsgBox "The card you have inserted does not appear to be a blank Master Key card."

```

```

    End If

```

```

End Sub

```

```

Private Sub ReaderEvent CardRemoved()

```

```

    ' Only allow them to remove the card without canceling

```

```

    ' when they are permitted to swap cards.

```

```

    If Not mvarOkToRemove Then

```

```

        Unload Me

```

```

    End If

```

frmMasterDuplicate - 6

End Sub

Private Sub txtTransport_Change()

 If ValidatePassword(txtTransport.Text) Then
 ' Only do this if we are on the transport frame.
 If FrameNumber = FRAME_TRANSPORT Then
 cmdNext.Enabled = False
 imgLock = imgLocked
 End If
 Else
 cmdNext.Enabled = True
 imgLock = imgUnlocked
 End If

End Sub

Private Sub txtCurrentMasterPass_Change()

 If ValidatePassword(txtCurrentMasterPass.Text) Then
 ' Only do this if we are on the transport frame.
 If FrameNumber = FRAME_MASTER_PASS Then
 cmdNext.Enabled = False
 imgMasterLock = imgLocked
 End If
 Else
 cmdNext.Enabled = True
 imgMasterLock = imgUnlocked
 End If

End Sub

O

frmMasterEdit - 1

Option Explicit

```
Const FRAME_TRANSPORT = 1
Const FRAME_USER_INFO = 2
Const FRAME_ADDKEYS = 3
Const FRAME_DONE = 5
```

```
Const FRAME_BORDER = 0
Const FRAME_OFF = -10000
Const FRAME_LEFT = 60
Const FRAME_TOP = 60
Const REQUIRED_PASSWORD_LEN = 8
Const AVAILABLE_KEY = "...(key slot available)..."
```

```
' Maximum length of a keyname for the validation proc.
Const MAX_KEY_LEN = 24
```

```
' Display variables
Private FrameNumber As Long
Private LastFrame As Long
' Special members
Private keys As Collection
Private RefKeys As New Collection
```

```
Private mvarNewKey As New CNewKey
Private mvarMasterCard As CSmartCard
Private WithEvents ReaderEvent As CReaderEngine
```

```
' Encryption API
Private Declare Function HashBuffer Lib "icell30.dll" (ByVal passphrase As String, ByVal phraselen As Long, ByVal hashval As Any) As Long
```

Private Sub Form_Load()

Dim x As Long

```
Set ReaderEvent = mvarReader
' Initialize the back and next keys to the startup state.
cmdBack.Enabled = False
cmdNext.Enabled = True
cmdFinish.Enabled = False
```

```
' Initialize the number of the last frame. The "done" frame
' is not counted, either.
```

```
LastFrame = Frame().Count - 2
```

```
' Set the first frame to be shown.
```

```
FrameNumber = 0
```

```
' Make sure all frames are hidden
```

```
For x = 0 To LastFrame
```

```
HideFrame x
```

```
Next
```

```
' Remove the borders
```

```
For x = 0 To FRAME_DONE
```

```
Frame(x).BorderStyle = FRAME_BORDER
```

```
Next
```

```
pb.Visible = False
```

```
' See if the Next button should be enabled
```

```
If FrameNumber = LastFrame Then cmdNext.Enabled = False
```

```
' Show the first frame
```

```
ShowFrame FrameNumber
```

End Sub

Private Sub Form_Unload(Cancel As Integer)

```
' Release the key collection to free resources.
```

```
Set keys = Nothing
```

```
Set RefKeys = Nothing
```

```
Set mvarMasterCard = Nothing
```

```
Set ReaderEvent = Nothing
```

```
' Re-enable main
```

frmMasterEdit - 2

frmMain.Enabled = True

End Sub

Private Sub cmdAddKey_Click()

' Allow the user to create a new key and add it to the list of defined keys.

Dim a As String

Dim t As String

Dim szBuffer As String * 22

Dim p As Integer

Dim x As Long

Dim KeyFields As Collection

Dim Structure As New CSmartCardFileStruct

' Create a file structure for reference

Structure.Load (GetFileStruct)

' Initialize the current key

mvarNewKey.KeyDate = Date

mvarNewKey.KeyName = ""

mvarNewKey.KeyValue = ""

' Show the key generator

Set frmGenerate.frmCaller = Me

frmGenerate.Show vbModal, Me

' Make sure a key was generated.

If mvarNewKey.KeyName <> "" Then

' A key was generated. SHA the return value

HashBuffer mvarNewKey.KeyValue, Len(mvarNewKey.KeyValue), szBuffer

NukeNulls szBuffer

' Now parse out 8 of the characters from the SHA result

For x = 1 To 8

' Pick a random position, 0 to 22 position in the string

Do

p = Int((22 * Rnd) + 1)

' Don't pick the same character position twice

Loop While InStr(1, t, Chr\$(p), vbBinaryCompare) > 0

' Save p in the string buffer so we know what we've already picked.

t = t & Chr\$(p)

' Pick up the indexed char

a = a & Mid\$(szBuffer, p, 1)

Next

' Assign the final 8 chars to the keyval

mvarNewKey.KeyValue = a

' Add the key name to the key collection

keys.Add mvarNewKey, mvarNewKey.KeyName

' Update the display

UpdateList

SetFrame FrameNumber

End If

End Sub

Private Function NukeNulls(szData As String)

' Remove all null chars from the string

Dim p As Integer

Do

p = InStr(1, szData, Chr\$(0), vbBinaryCompare)

If p > 0 Then

Mid\$(szData, p, 1) = Chr\$(Int((254 * Rnd) + 1))

MsgBox "Nailed a null"

End If

Loop While p > 0

End Function

Public Property Get CurrentKey() As CNewKey

' Return a reference to this object's current key

Set CurrentKey = mvarNewKey

End Property

```
Private Sub UpdateList()
```

```
    ' Update the smart card display on the form.
```

```
    Dim n As Long
```

```
    Dim x As Long
```

```
    ' Clear the list
```

```
    lstKeys.Clear
```

```
    ' Add the keys in the collection
```

```
    For Each mvarNewKey In keys
```

```
        lstKeys.AddItem mvarNewKey.KeyName
```

```
    Next
```

```
    ' Fill in the rest of the display
```

```
    n = GetAllowedKeyCount
```

```
    For x = keys.Count To n - 1
```

```
        lstKeys.AddItem AVAILABLE_KEY
```

```
    Next
```

```
    ' Enable or disable the add key button
```

```
    If AvailableKeys = 0 Then
```

```
        cmdAddKey.Enabled = False
```

```
    Else
```

```
        cmdAddKey.Enabled = True
```

```
    End If
```

```
End Sub
```

```
Private Sub cmdCancel_Click()
```

```
    MsgBox "You must remove the Master card from the reader before the wizard can be canceled."
```

```
End Sub
```

```
Private Sub cmdDeleteKey_Click()
```

```
    ' Allow the user to delete an existing key.
```

```
    Dim key As New CNewKey
```

```
    Dim nRet As Long
```

```
    ' Make sure the user has selected a valid key to delete
```

```
    If lstKeys.List(lstKeys.ListIndex) <> AVAILABLE_KEY Then
```

```
        ' Retrieve the key to delete.
```

```
        For Each key In keys
```

```
            If key.KeyName = lstKeys.List(lstKeys.ListIndex) Then
```

```
                ' Make sure this is not an old key
```

```
                If Not IsOld(key) Then
```

```
                    ' Confirm deletion
```

```
                    nRet = MsgBox("Are you sure you wish to delete the key " & Chr$(34) & key.KeyName & Chr$(34) & "?", vbYesNo)
```

```
                    If nRet = vbYes Then
```

```
                        ' Nail it.
```

```
                        keys.Remove (key.KeyName)
```

```
                        ' Update the display
```

```
                        UpdateList
```

```
                        SetFrame FrameNumber
```

```
                        Exit For
```

```
                    End If
```

```
                End If
```

```
            End If
```

```
        Next
```

```
    End If
```

```
End Sub
```

```
Private Sub cmdEdit_Click()
```

```
    ' Allow the user to edit the current key.
```

```
    EditKey lstKeys.ListIndex
```

```
End Sub
```

```
Private Sub cmdFinish_Click()
```

```
    ' Write the keys to the master card, and close the card.
```

```
Dim nRet As Long
```

```
' One last check before writing the keys.
```

```
nRet = MsgBox("Are you sure you want to write the master key card now?", vbYesNo)
```

```
If nRet = vbYes Then
    frameStatus.Visible = True
    BurnBabyBurn 1
End If
```

```
End Sub
```

```
Private Sub BurnBabyBurn(nStructID As Long)
```

```
' Dispatch the correct burn function
```

```
Select Case nStructID
```

```
Case 1
```

```
' Dispatch
```

```
BurnStructOne
```

```
Case Else
```

```
End Select
```

```
End Sub
```

```
Private Sub BurnStructOne()
```

```
Dim Header As New CCardHeader
```

```
Dim Card As New CSmartCard
```

```
Dim nRet As Long
```

```
Dim a As String
```

```
a = GetMasterIDString
```

```
' Mark it as locked
```

```
Mid$(a, 2, 1) = Chr$(1)
```

```
Header.MasterIDString = a
```

```
' Header.TryCount = "0"
```

```
Header.UserName = txtUserName
```

```
Header.UserPassword = txtPassword
```

```
Header.Reserved = Chr$(0)
```

```
' Header.IntelligardID = GetNewID
```

```
nRet = WriteHeader(Card, Header)
```

```
nRet = WriteKeys(nRet, Card, keys)
```

```
If Card.WriteCard(pbl) <> 0 Then
```

```
' Error in key creation.
```

```
MsgBox "There has been an error during master key creation. Please contact your system administrator."
```

```
Else
```

```
' Success. Leave them no choice but to remove the smart card.
```

```
Me.cmdNext = False
```

```
Me.cmdBack.Enabled = False
```

```
Me.cmdCancel.Enabled = False
```

```
Me.cmdFinish.Enabled = False
```

```
ShowFrame FRAME_DONE
```

```
End If
```

```
End Sub
```

```
Private Sub cmdReset_Click()
```

```
' Undo whatever user info changes were made by reloading the info
```

```
txtUserName = GetUserName
```

```
txtPassword = GetPassword
```

```
txtVerify = txtPassword
```

```
End Sub
```

```
Private Sub cmdBack_Click()
```

```
' Hide the current frame
```

frmMasterEdit - 5

HideFrame FrameNumber

' Go back to the last frame.

FrameNumber = FrameNumber - 1

' Show the new frame

ShowFrame FrameNumber

If FrameNumber = 0 Then

' At the first frame, so disable the "back" button.

cmdBack.Enabled = False

End If

If FrameNumber < LastFrame Then

' Not at the last frame. Disable the finish button.

cmdFinish.Enabled = False

End If

' If you're going backward, you're coming from an enabled frame, so

' make sure the next button is enabled, in case it was the last frame.

If FrameNumber <> FRAME_TRANSPORT Then

cmdNext.Enabled = True

End If

End Sub

Private Sub cmdNext Click()

' Check the current status of the data before moving on.

If Not Validate(FrameNumber) Then

' Hide the current frame

HideFrame FrameNumber

' Advance to the next frame.

FrameNumber = FrameNumber + 1

' Show the next frame

ShowFrame FrameNumber

' Check if the frame should be disabled.

If FrameNumber = LastFrame Then

' At the last frame, so disable the "next" button, enable finish button

cmdNext.Enabled = False

cmdFinish.Enabled = True

End If

' Check if the back button should be enabled

If FrameNumber > 0 Then

cmdBack.Enabled = True

End If

End If

End Sub

Private Sub HideFrame(n As Long)

' Hide the frame passed in N

Frame(n).Left = FRAME_OFF

End Sub

Private Sub ShowFrame(n As Long)

' Hide the frame passed in N

Frame(n).Left = FRAME_LEFT

Frame(n).Top = FRAME_TOP

Frame(n).Width = Frame(0).Width

Frame(n).Height = Frame(0).Height

' Set special case conditions here.

SetFrame n

End Sub

Private Function Validate(FrameNumber As Long) As Boolean

' Apply validation rules to data on frame passed in.

' Returns true if the data is not acceptable.

Select Case FrameNumber

Case FRAME_TRANSPORT

' Password frame. Validate the password.

Validate = ValidatePassword

If Validate = False Then

' Don't let a valid transport password remain exposed - even for cut and paste.

'txtTransport = ""

End If

Case FRAME_USER_INFO

' Check the new password and username

Validate = CheckNewPass

Case Else

End Select

End Function

Private Function CheckNewPass() As Boolean

' Returns false if the data on the new password frame is okay.

If Me.txtUserName = "" Then

' Sorry, gotta have a username.

CheckNewPass = True

MsgBox "You must enter a username for this card."

Exit Function

End If

If Me.txtPassword <> Me.txtVerify Then

' Sorry, passwords don't match

CheckNewPass = True

MsgBox "The validation password does not match the new password. Please try again."

Exit Function

End If

If Me.txtPassword = "" Then

' Duh. Blank Password not permitted.

CheckNewPass = True

MsgBox "You must enter a password to be used for this card."

Exit Function

End If

If Len(Me.txtPassword) < REQUIRED_PASSWORD_LEN Then

' Sorry, gotta have at least x number of characters.

CheckNewPass = True

MsgBox "The new password must be at least " & REQUIRED_PASSWORD_LEN & " characters long."

Exit Function

End If

End Function

Private Function setFrame(n As Long)

' Set up special conditions for the frame being shown.

Dim szKeys As String

Dim szSlots As String

Dim t As Long

Select Case n

Case FRAME_TRANSPORT

cmdNext.Enabled = False

imgLock = imgLocked

txtTransport = ""

txtTransport.SetFocus

Case FRAME_USER_INFO

' Set up the current user info

If txtUserName = "" Then txtUserName = GetUserName

If txtPassword = "" Then txtPassword = GetPassword: txtVerify = txtPassword

Case FRAME_ADDKEYS

```

    lblKeyCount = "Loading keys..."
    pb.Visible = True
    ' Populate the listbox with the keys on the card, if keycount is 0
    LoadEmptyKeys pb
    pb.Visible = False
    ' Tell the user how many keys they have left.
    lblKeyCount = KeyUpdateString
    ' And that no keys are selected on the list
    lstKeys.ListIndex = -1
    ' Make sure the key edit button is disabled on load.
    cmdEdit.Enabled = False
    cmdDeleteKey.Enabled = False
    ' Set the state of the "next" button to require at least one key to be defined.
    If AvailableKeys = lstKeys.ListCount Then
        cmdNext.Enabled = False
    Else
        cmdNext.Enabled = True
    End If
    If cmdAddKey.Enabled Then
        ' Default the add button
        Me.cmdAddKey.SetFocus
    End If

```

Case LastFrame

Case Else

```

    ' Do nothing

```

```

End Select

```

End Function

```

Private Function ValidatePassword() As Boolean

```

```

    ' Compare the user's password to the transport password on the card.

```

```

    ' Returns false if the password matches

```

```

    Static szPass As String

```

```

    If szPass = "" Then

```

```

        szPass = GetPassword

```

```

    End If

```

```

    If txtTransport.Text = szPass Then

```

```

        ' Password matches.

```

```

        ValidatePassword = False

```

```

    Else

```

```

        ' Bzzz. Sorry, please drive through.

```

```

        ValidatePassword = True

```

```

    End If

```

End Function

```

Private Function AvailableKeys()

```

```

    ' Returns the number of keys the user has left to define on their card.

```

```

    Dim x As Long

```

```

    Dim c As Long

```

```

    ' Loop through the keylist and count the number of open slots.

```

```

    For x = 0 To lstKeys.ListCount - 1

```

```

        If lstKeys.List(x) = AVAILABLE_KEY Then

```

```

            c = c + 1

```

```

        End If

```

```

    Next

```

```

    AvailableKeys = c

```

End Function

```

Private Function LoadEmptyKeys(Optional pb As ProgressBar)

```

```

    ' Fill the list box with the number of available spaces they have for keys.

```

```

    Dim mvarMasterCard As CSmartCard

```

```

    Dim key As New CNewKey

```

' Only do this once.

If RefKeys.Count = 0 Then

' Instantiate a smart card data struct
Set mvarMasterCard = New CSmartCard
' Load it.
mvarMasterCard.ReadCard pb
Set keys = mvarMasterCard.keys
' Display the keys currently on this card
UpdateList
' Create an internal copy of the keys for reference
For Each key In keys
 RefKeys.Add key
Next

End If

End Function

Private Function KeyUpdateString()

' Returns a string telling the user how many keys they have left on the card.

Dim szKeys As String
Dim szSlots As String

If lstKeys.ListCount = 1 Then
 szKeys = " key."
Else
 szKeys = " keys."
End If

If AvailableKeys = 1 Then
 szSlots = "slot"
Else
 szSlots = "slots"
End If

KeyUpdateString = "The current card can hold " & lstKeys.ListCount & szKeys & " You have " & AvailableKeys & " open " & szSlots & " available."

End Function

Private Sub EditKey(index As Long)

' Allow the user to edit an existing key.

Dim key As New CNewKey
Dim t As New CNewKey
Dim bNoWay As Boolean
Dim bFound As Boolean

' Retrieve the key to edit.

For Each t In keys
 If t.KeyName = lstKeys.List(index) Then
 bFound = True
 Set key = keys(t.KeyName)
 End If
Next

If Not bFound Then

' This is a major error. How can the key be on the list, but not in the collection? Oops

MsgBox "There has been an internal error with the Intelligard Card Administrator. Please contact your system administrator."

Else

' Found the key that the user has selected. Check if edits are allowed

If IsOld(key) Then

 MsgBox "The key you have selected may not be edited."

Else

' This is a new key, it's okay to edit.

 EditKeyName key

End If

End If

End Sub

Private Function IsOld(key As CNewKey)

' Returns true if the key passed in exists on the master card.

Dim t As CNewKey

For Each t In RefKeys

If t.KeyName = key.KeyName Then

' The key exists already. No edits. Sorry.

IsOld = True

Exit For

End If

Next

End Function

Private Sub EditKeyName(key As CNewKey)

' Edit the name of the key passed in.

Dim Message As String

Dim Title As String

Dim Default As String

Dim szRet As String

Message = "Enter the name for the key."

Title = "Edit Key"

Default = key.KeyName

' Display message, title, and default value.

szRet = InputBox(Message, Title, Default)

If Not ValidateKeyName(szRet) Then

If szRet <> "" Then

' Remove the old key from the key collection

keys.Remove (key.KeyName)

' Assign the new key name.

key.KeyName = szRet

' Add the new key to the collection

keys.Add key, key.KeyName

' Update the display

UpdateList

SetFrame FrameNumber

End If

Else

' The name they input was not unique. Bitch at them

MsgBox "Invalid key name. Key names must be unique, and no longer than " & "16" & " characters."

End If

End Sub

Public Function ValidateKeyName(szNewName As String) As Boolean

' Check that the key name provided is unique in the collection.

Dim key As New CNewKey

For Each key In keys

If UCase\$(key.KeyName) = UCase\$(szNewName) Then

' No good. They already have this key name.

ValidateKeyName = True

End If

Next

If szNewName = "" Then ValidateKeyName = True

If Len(szNewName) > MAX_KEY_LEN Then ValidateKeyName = True

End Function

Private Sub lstKeys_Click()

' Update the available commands.

If lstKeys.List(lstKeys.ListIndex) = AVAILABLE_KEY Then

' This is an open slot. Not much can be done.

```
Me.cmdDeleteKey.Enabled = False
```

```
Me.cmdEdit.Enabled = False
```

```
Else
```

```
' Make sure this key is not an existing key.
```

```
Dim key As CNewKey
```

```
Dim bFound As Boolean
```

```
For Each key In RefKeys
```

```
    If key.KeyName = lstKeys.List(lstKeys.ListIndex) Then
```

```
        bFound = True
```

```
        Exit For
```

```
    End If
```

```
Next
```

```
' Only enable edit keys if this is not an existing key
```

```
If Not bFound Then
```

```
    cmdDeleteKey.Enabled = True
```

```
    cmdEdit.Enabled = True.
```

```
Else
```

```
    cmdDeleteKey.Enabled = False
```

```
    cmdEdit.Enabled = False
```

```
End If
```

```
End If
```

```
End Sub
```

```
Private Sub lstKeys_DblClick()
```

```
    If lstKeys.List(lstKeys.ListIndex) = AVAILABLE_KEY Then
```

```
        ' The user double clicked an available key slot. Let them add a key  
        ' but only to the next available slot, not the one they clicked.
```

```
        cmdAddKey_Click
```

```
    Else
```

```
        ' Allow the user to edit the current key.
```

```
        EditKey lstKeys.ListIndex
```

```
    End If
```

```
End Sub
```

```
Private Sub ReaderEvent_CardRemoved()
```

```
    Unload Me
```

```
End Sub
```

```
Private Sub txtTransport_Change()
```

```
    If ValidatePassword Then
```

```
        ' Only do this if we are on the transport frame.
```

```
        If FrameNumber = FRAME_TRANSPORT Then
```

```
            cmdNext.Enabled = False
```

```
            imgLock = imgLocked
```

```
        End If
```

```
    Else
```

```
        cmdNext.Enabled = True
```

```
        imgLock = imgUnlocked
```

```
    End If
```

```
End Sub
```

frmMasterWizard - 1

Option Explicit

Const FRAME_TRANSPORT = 1
Const FRAME_ADDKEYS = 3
Const FRAME_DONE = 5

Const FRAME_BORDER = 0
Const FRAME_OFF = -10000
Const FRAME_LEFT = 60
Const FRAME_TOP = 60
Const REQUIRED_PASSWORD_LEN = 6
Const AVAILABLE_KEY = "... (key slot available) ..."

' Maximum length of a keyname for the validation proc.
Const MAX_KEY_LEN = 24

' Display variables

Private FrameNumber As Long
Private LastFrame As Long

' Special members

Private keys As New Collection

Private mvarNewKey As New CNewKey

' Encryption API

Private Declare Function HashBuffer Lib "icell30.dll" (ByVal passphrase As String, ByVal phraselen As Long, ByVal hashval As Any) As Long

Private WithEvents ReaderEvent As CReaderEngine

Private Sub Form_Load()

Dim x As Long

' Initialize a reference to the card reader engine

Set ReaderEvent = mvarReader

' Initialize the back and next keys to the startup state.

cmdBack.Enabled = False

cmdNext.Enabled = True

cmdFinish.Enabled = False

' Initialize the number of the last frame. The "done" frame
' is not counted, either.

LastFrame = Frame().Count - 2

' Set the first frame to be shown.

FrameNumber = 0

' Make sure all frames are hidden

For x = 0 To LastFrame

HideFrame x

Next

' Remove the borders

For x = 0 To FRAME_DONE

Frame(x).BorderStyle = FRAME_BORDER

Next

' See if the Next button should be enabled

If FrameNumber = LastFrame Then cmdNext.Enabled = False

' Show the first frame

ShowFrame FrameNumber

End Sub

Private Sub Form_Unload(Cancel As Integer)

' Release the key collection to free resources.

Set keys = Nothing

Set ReaderEvent = Nothing

frmMain.Enabled = True

End Sub

Private Sub cmdAddKey_Click()

' Allow the user to create a new key and add it to the list of defined keys.

Dim a As String

Dim t As String

Dim szBuffer As String * 22

```

Dim p As Integer
Dim x As Long
Dim KeyFields As Collection
Dim Structure As New CSmartCardFileStruct

```

```

' Create a file structure for reference

```

```

Structure.Load (GetFileStruct)

```

```

' Initialize the current key

```

```

mvarNewKey.KeyDate = Date

```

```

mvarNewKey.KeyName = ""

```

```

mvarNewKey.KeyValue = ""

```

```

' Show the key generator

```

```

Set frmGenerate.frmCaller = Me

```

```

frmGenerate.Show vbModal, Me

```

```

' Make sure a key was generated.

```

```

If mvarNewKey.KeyName <> "" Then

```

```

    ' A key was generated. SHA the return value

```

```

    HashBuffer mvarNewKey.KeyValue, Len(mvarNewKey.KeyValue), szBuffer

```

```

    NukeNulls szBuffer

```

```

    ' Now parse out 8 of the characters from the SHA result

```

```

    For x = 1 To 8

```

```

        ' Pick a random position, 0 to 22 position in the string

```

```

        Do

```

```

            p = Int((22 * Rnd) + 1)

```

```

            ' Don't pick the same character position twice

```

```

            Loop While InStr(1, t, Chr$(p), vbBinaryCompare) > 0

```

```

            ' Save p in the string buffer so we know what we've already picked.

```

```

            t = t & Chr$(p)

```

```

            ' Pick up the indexed char

```

```

            a = a & Mid$(szBuffer, p, 1)

```

```

        Next

```

```

        Dim hFile As Long

```

```

        Dim sig As String

```

```

        hFile = FreeFile

```

```

        Open App.Path & "\Yuval-pw.cps" For Binary As hFile

```

```

        sig = Space$(LOF(hFile))

```

```

        Get #hFile, 1, sig

```

```

        Close hFile

```

```

        ' Assign the final 8 chars to the keyval

```

```

        mvarNewKey.KeyValue = sig

```

```

        ' Add the key name to the key collection

```

```

        keys.Add mvarNewKey, mvarNewKey.KeyName

```

```

        ' Update the display

```

```

        UpdateList

```

```

        SetFrame FrameNumber

```

```

    End If

```

```

End Sub

```

```

Private Function NukeNulls(szData As String)

```

```

    ' Remove all null chars from the string

```

```

    Dim p As Integer

```

```

    Do

```

```

        p = InStr(1, szData, Chr$(0), vbBinaryCompare)

```

```

        If p > 0 Then

```

```

            Mid$(szData, p, 1) = Chr$(Int((254 * Rnd) + 1))

```

```

            MsgBox "Nailed a null"

```

```

        End If

```

```

    Loop While p > 0

```

```

End Function

```

```

Public Property Get CurrentKey() As CNewKey

```

```

    ' Return a reference to this object's current key

```

```

    Set CurrentKey = mvarNewKey

```

```

End Property

```

```

'Private Function BlankKey() As CNewKey

```

```
' Instantiate and initialize a new key instance
```

```
    Dim KeyFields As New Collection
    Dim Field As CField
```

```
    ' Create a file structure for reference
    Dim Structure As New CSmartCardFileStruct
    Structure.Load (GetFileStruct)
```

```
    ' Initialize the current key
    mvarNewKey.KeyDate = Date
    mvarNewKey.KeyName = ""
    mvarNewKey.KeyValue = ""
```

```
    ' Set a ref to the key's fields collection
    Set KeyFields = mvarNewKey.KeyFields
```

```
    ' Init the key fields collection
    Set Field = Structure.Fields("KeyID")
    KeyFields.Add Field
```

```
    KeyFields.Add Structure.Fields("KeyDate")
    KeyFields.Add Structure.Fields("KeyValue")
```

```
'End Function
```

```
Private Sub UpdateList()
```

```
    ' Update the smart card display on the form.
```

```
    Dim n As Long
    Dim x As Long
```

```
    ' Clear the list
    lstKeys.Clear
```

```
    ' Add the keys in the collection
    For Each mvarNewKey In keys
        lstKeys.AddItem mvarNewKey.KeyName
    Next
```

```
    ' Fill in the rest of the display
    n = GetAllowedKeyCount
    For x = keys.Count To n - 1
        lstKeys.AddItem AVAILABLE_KEY
    Next
```

```
    ' Enable or disable the add key button
    If AvailableKeys = 0 Then
        cmdAddKey.Enabled = False
    Else
        cmdAddKey.Enabled = True
    End If
```

```
End Sub
```

```
Private Sub cmdCancel_Click()
```

```
    MsgBox "You must remove the Master card from the reader before the wizard can be canceled."
    Unload Me
```

```
End Sub
```

```
Private Sub cmdDeleteKey_Click()
```

```
    ' Allow the user to delete an existing key.
```

```
    Dim key As New CNewKey
    Dim nRet As Long
```

```
    ' Make sure the user has selected a valid key to delete
    If lstKeys.List(lstKeys.ListIndex) <> AVAILABLE_KEY Then
```

```
        ' Retrieve the key to edit.
```

```
        For Each key In keys
            If key.KeyName = lstKeys.List(lstKeys.ListIndex) Then
```

```
                ' Confirm deletion
```

```
                nRet = MsgBox("Are you sure you wish to delete the key " & Chr$(34) & key.KeyName & Chr$(34) & "?", vbYesNo)
```

```

    If nRet = vbYes Then
        ' Nail it.
        keys.Remove (key.KeyName)
        ' Update the display
        UpdateList
        SetFrame FrameNumber
    End If

```

```

End If
Next

```

```

End If

```

```

End Sub

```

```

Private Sub cmdEdit_Click()

```

```

    ' Allow the user to edit the current key.
    EditKey lstKeys.ListIndex

```

```

End Sub

```

```

Private Sub cmdFinish_Click()

```

```

    ' Write the keys to the master card, and close the card.

```

```

    Dim nRet As Long

```

```

    ' One last check before writing the keys.

```

```

    nRet = MsgBox("Are you sure you want to write the master key card now?", vbYesNo)

```

```

    If nRet = vbYes Then

```

```

        frameStatus.Visible = True

```

```

        BurnBabyBurn 1

```

```

    End If

```

```

End Sub

```

```

Private Sub BurnBabyBurn(nStructID As Long)

```

```

    ' Dispatch the correct burn function

```

```

    Select Case nStructID

```

```

        Case 1

```

```

            ' Dispatch

```

```

            BurnStructOne

```

```

        Case Else

```

```

    End Select

```

```

End Sub

```

```

Private Sub BurnStructOne()

```

```

    Dim Header As New CCardHeader

```

```

    Dim Card As New CSmartCard

```

```

    Dim nRet As Long

```

```

    Dim a As String

```

```

    a = GetMasterIDString

```

```

    ' Mark it as locked

```

```

    Mid$(a, 2, 1) = Chr$(1)

```

```

    Header.MasterIDString = a

```

```

    ' Set the master key string locked.

```

```

    Header.TryCount = "0"

```

```

    ' Clear the fail counter

```

```

    Header.UserName = txtUserName

```

```

    ' Assign the user name

```

```

    Header.UserPassword = txtPassword

```

```

    ' Assign the admin password

```

```

    Header.Reserved = Chr$(0)

```

```

    Header.IntelligardID = GetNewID

```

```

    ' Get the serial number

```

```

    nRet = WriteHeader(Card, Header)

```

```

    nRet = WriteKeys(nRet, Card, keys)

```

```

    If Card.WriteCard(pb1) <> 0 Then

```

```

        ' Error in key creation.

```

```

        MsgBox "There has been an error during master key creation. Please contact your system administrator."

```

```

    Else

```

```

' Success. Leave them no choice but to remove the smart card.
Me.cmdNext = False
Me.cmdBack.Enabled = False
Me.cmdCancel.Enabled = False
Me.cmdFinish.Enabled = False
ShowFrame FRAME_DONE
End If

```

End Sub

Private Sub cmdBack_Click()

```

' Hide the current frame
HideFrame FrameNumber
' Go back to the last frame.
FrameNumber = FrameNumber - 1

' Show the new frame
ShowFrame FrameNumber

If FrameNumber = 0 Then
' At the first frame, so disable the "back" button.
cmdBack.Enabled = False
End If
If FrameNumber < LastFrame Then
' Not at the last frame. Disable the finish button.
cmdFinish.Enabled = False
End If

' If you're going backward, you're coming from an enabled frame, so
' make sure the next button is enabled, in case it was the last frame.
If FrameNumber <> FRAME_TRANSPORT Then
cmdNext.Enabled = True
End If

```

End Sub

Private Sub cmdNext_Click()

```

' Check the current status of the data before moving on.
If Not Validate(FrameNumber) Then

' Hide the current frame
HideFrame FrameNumber
' Advance to the next frame
FrameNumber = FrameNumber + 1
' Show the next frame
ShowFrame FrameNumber
' Check if the frame should be disabled.
If FrameNumber = LastFrame Then
' At the last frame, so disable the "next" button, enable finish button
cmdNext.Enabled = False
cmdFinish.Enabled = True
End If
' Check if the back button should be enabled
If FrameNumber > 0 Then
cmdBack.Enabled = True
End If

End If

```

End Sub

Private Sub HideFrame(n As Long)

```

' Hide the frame passed in N

Frame(n).Left = FRAME_OFF

```

End Sub

Private Sub ShowFrame(n As Long)

```

' Hide the frame passed in N

```

```

Frame(n).Left = FRAME_LEFT
Frame(n).Top = FRAME_TOP
Frame(n).Width = Frame(0).Width
Frame(n).Height = Frame(0).Height

```

```

' Set special case conditions here.
SetFrame n

```

```

End Sub

```

```

Private Function Validate(FrameNumber As Long) As Boolean
' Apply validation rules to data on frame passed in.
' Returns true if the data is not acceptable.

```

```

Select Case FrameNumber
Case 1
' Password frame. Validate the password.
Validate = ValidatePassword
If Validate = False Then
' Don't let a valid transport password remain exposed - even for cut and paste.
'txtTransport = ""
End If
Case 2
' Check the new password and username
Validate = CheckNewPass
Case Else
End Select

```

```

End Function

```

```

Private Function CheckNewPass() As Boolean
' Returns false if the data on the new password frame is okay.

```

```

If Me.txtUserName = "" Then
' Sorry, gotta have a username.
CheckNewPass = True
MsgBox "You must enter a username for this card."
Exit Function
End If

```

```

If Me.txtPassword <> Me.txtVerify Then
' Sorry, passwords don't match
CheckNewPass = True
MsgBox "The validation password does not match the new password. Please try again."
Exit Function
End If

```

```

If Me.txtPassword = "" Then
' Duh. Blank Password not permitted.
CheckNewPass = True
MsgBox "You must enter a password to be used for this card."
Exit Function
End If

```

```

If Len(Me.txtPassword) < REQUIRED_PASSWORD_LEN Then
' Sorry, gotta have at least x number of characters.
CheckNewPass = True
MsgBox "The new password must be at least " & REQUIRED_PASSWORD_LEN & " characters long."
Exit Function

```

```

End If

```

```

End Function

```

```

Private Function SetFrame(n As Long)
' Set up special conditions for the frame being shown.
Dim szKeys As String
Dim szSlots As String
Dim t As Long

```


Select Case n

Case FRAME_TRANSPORT

cmdNext.Enabled = False

imgLock = imgLocked

txtTransport = ""

txtTransport.SetFocus

Case FRAME_ADDKEYS

' Populate the listbox with the keys on the card.

LoadEmptyKeys

' Tell the user how many keys they have left.

lblKeyCount = KeyUpdateString

' And that no keys are selected on the list

lstKeys.ListIndex = -1

' Make sure the key edit button is disabled on load.

cmdEdit.Enabled = False

' Set the state of the "next" button to require at least one key to be defined.

If AvailableKeys = lstKeys.ListCount Then

cmdNext.Enabled = False

Else

cmdNext.Enabled = True

End If

If cmdAddKey.Enabled Then

' Default the add button

Me.cmdAddKey.SetFocus

End If

Case LastFrame

Case Else

' Do nothing

End Select

Function

Private Function ValidatePassword() As Boolean

' Compare the user's password to the transport password on the card.

' Returns false if the password matches

Static szPass As String

If szPass = "" Then

szPass = GetPassword

End If

If txtTransport.Text = szPass Then

' Password matches.

ValidatePassword = False

Else

' Bzzz. Sorry, please drive through.

ValidatePassword = True

End If

End Function

Private Function AvailableKeys()

' Returns the number of keys the user has left to define on their card.

Dim x As Long

Dim c As Long

' Loop through the keylist and count the number of open slots.

For x = 0 To lstKeys.ListCount - 1

If lstKeys.List(x) = AVAILABLE_KEY Then

c = c + 1

End If

Next

AvailableKeys = c

End Function

Private Function LoadEmptyKeys()

' Fill the list box with the number of available spaces they have for keys.

```
Dim n As Long
Dim x As Long
```

```
' Only do this the first time the list is displayed.
```

```
If lstKeys.ListCount = 0 Then
```

```
' Disable the delete keys command. There are no keys to delete.
```

```
Me.cmdDeleteKey.Enabled = False
```

```
' Get the number of keys the user is allowed
```

```
n = GetAllowedKeyCount
```

```
' Make sure they are allowed any - i.e. is this a valid blank master card?
```

```
If n > 0 Then
```

```
' Loop the number of keys
```

```
For x = 1 To n
```

```
' Add the available number of keys as placeholders
```

```
lstKeys.AddItem AVAILABLE_KEY
```

```
Next
```

```
End If
```

```
End If
```

```
End Function
```

```
Private Function KeyUpdateString()
```

```
' Returns a string telling the user how many keys they have left on the card.
```

```
Dim szKeys As String
```

```
Dim szSlots As String
```

```
If lstKeys.ListCount = 1 Then
```

```
szKeys = " key."
```

```
Else
```

```
szKeys = " keys."
```

```
End If
```

```
If AvailableKeys = 1 Then
```

```
szSlots = "slot"
```

```
Else
```

```
szSlots = "slots"
```

```
End If
```

```
KeyUpdateString = "The current card can hold " & lstKeys.ListCount & szKeys & " You have " &  
AvailableKeys & " open " & szSlots & " available."
```

```
End Function
```

```
Private Sub EditKey(index As Long)
```

```
' Allow the user to edit an existing key.
```

```
Dim key As New CNewKey
```

```
Dim t As New CNewKey
```

```
Dim bFound As Boolean
```

```
' Retrieve the key to edit.
```

```
For Each t In keys
```

```
If t.KeyName = lstKeys.List(index) Then
```

```
bFound = True
```

```
Set key = keys(t.KeyName)
```

```
End If
```

```
Next
```

```
If Not bFound Then
```

```
' This is a major error. How can the key be on the list, but not in the collection? Oops
```

```
MsgBox "There has been an internal error with the Intelligard Card Administrator. Please  
contact your system administrator."
```

```
Else
```

```
' Found the key that the user has selected. Edit it.
```

```
EditKeyName key
```

```
End If
```

```
End Sub
```

```
Private Sub EditKeyName(key As CNewKey)
```

```
' Edit the name of the key passed in.
```

```
    Dim Message As String
```

```
    Dim Title As String
```

```
    Dim Default As String
```

```
    Dim szRet As String
```

```
    Message = "Enter the name for the key."
```

```
    Title = "Edit Key"
```

```
    Default = key.KeyName
```

```
    ' Display message, title, and default value.
```

```
    szRet = InputBox(Message, Title, Default)
```

```
    If Not ValidateKeyName(szRet) Then
```

```
        If szRet <> "" Then
```

```
            ' Remove the old key from the key collection
```

```
            keys.Remove (key.KeyName)
```

```
            ' Assign the new key name.
```

```
            key.KeyName = szRet
```

```
            ' Add the new key to the collection
```

```
            keys.Add key, key.KeyName
```

```
            ' Update the display
```

```
            UpdateList
```

```
            SetFrame FrameNumber
```

```
        End If
```

```
    Else
```

```
        ' The name they input was not unique. Bitch at them
```

```
        MsgBox "Invalid key name. Key names must be unique, and no longer than " & "16" & " characters."
```

```
    End If
```

```
End Sub
```

```
Public Function ValidateKeyName(szNewName As String) As Boolean
```

```
' Check that the key name provided is unique in the collection.
```

```
    Dim key As New CNewKey
```

```
    For Each key In keys
```

```
        If UCase$(key.KeyName) = UCase$(szNewName) Then
```

```
            ' No good. They already have this key name.
```

```
            ValidateKeyName = True
```

```
            Exit For
```

```
        End If
```

```
    Next
```

```
    ' Check that it is not a reserved key name
```

```
    If (UCase$(szNewName) = "CODE BOOK") Or (UCase$(szNewName) = "CODEBOOK") Then
```

```
        ValidateKeyName = True
```

```
    End If
```

```
    ' Blank
```

```
    If szNewName = "" Then ValidateKeyName = True
```

```
    ' Or too long
```

```
    If Len(szNewName) > MAX_KEY_LEN Then ValidateKeyName = True
```

```
End Function
```

```
Private Sub lstKeys_Click()
```

```
' Update the available commands.
```

```
    If lstKeys.List(lstKeys.ListIndex) = AVAILABLE_KEY Then
```

```
        ' This is an open slot. Not much can be done.
```

```
        Me.cmdDeleteKey.Enabled = False
```

```
        Me.cmdEdit.Enabled = False
```

```
    Else
```

```
        Me.cmdDeleteKey.Enabled = True
```

```
        Me.cmdEdit.Enabled = True
```

```
    End If
```

```
End Sub
```

```
Private Sub lstKeys_DblClick()
```

```
    If lstKeys.List(lstKeys.ListIndex) = AVAILABLE_KEY Then
```

' The user double clicked an available key slot. Let them add a key
' but only to the next available slot, not the one they clicked.

cmdAddKey_Click

Else

' Allow the user to edit the current key.

EditKey lstKeys.ListIndex

End If

End Sub

Private Sub ReaderEvent_CardRemoved()

Unload Me

End Sub

Private Sub txtTransport_Change()

If ValidatePassword Then

' Only do this if we are on the transport frame.

If FrameNumber = FRAME_TRANSPORT Then

cmdNext.Enabled = False

imgLock = imgLocked

End If

Else

cmdNext.Enabled = True

imgLock = imgUnlocked

End If

End Sub

frmServer - 1

Public WithEvents ReaderEvent As CReaderEngine

Private Sub Form_GotFocus()

' Blow off "can't show modal form..." errors.

On Error Resume Next

Dim t As New CWindowServices

t.SetForeground Me.hWnd

t.SetWinOnTop Me.hWnd, True

End Sub

Private Sub ReaderEvent_CardRemoved()

Unload Me

End Sub

frmSplash - 1

Option Explicit

Private Sub Form_DblClick()

Unload Me

End Sub

Private Sub Form_Load()

lblVersion.Caption = "Version " & App.Major & "." & App.Minor & "." & App.Revision

lblProductName.Caption = "IntelliGard Smart Card Administrator"

tmrSplash.Enabled = True

' Make sure we're on top.

Dim t As New CWindowServices

t.SetForeground Me.hWnd

End Sub

Private Sub tmrSplash_Timer()

tmrSplash.Enabled = False

Unload Me

End Sub

modCardInterface - 1

Option Explicit

Const DATA_WIDTH = 4

' Ken - define this constant.

Const READER_BUSY = 1

Public Const TYPE_MASTER = "M"

Public Const TYPE_USER = "U"

Public Const TYPE_SYSADMIN = "A"

Public Const TYPE_BLANK = "B"

' Put all card types in a string for later reference.

Public Const CARD_TYPES = TYPE_MASTER & TYPE_USER & TYPE_SYSADMIN & TYPE_BLANK

Public Function AttemptBoot(Optional szKeyFileName As String) As String

Dim nRet As Long

nRet = CardOn(szKeyFileName)

If nRet <> 0 Then

' Error occurred during initialization of the card

AttemptBoot = "Unable to initialize the smart card."

End If

End Function

Public Function InitializeCard() As Boolean

Dim szTemp As String

szTemp = AttemptSelectRoot

If szTemp = "" Then szTemp = AttemptSelectEF

If szTemp = "" Then szTemp = AttemptVerify

If szTemp <> "" Then

MsgBox szTemp

' Return the failure.

InitializeCard = True

End If

End Function

Private Function AttemptSelectRoot() As String

Dim nRet As Long

Dim szTemp As String

' Create string from file number

szTemp = Chr\$(&H3D) + Chr\$(&H0)

nRet = SelectRootFile(szTemp)

If nRet <> 0 Then

' Error - failed to select root file

AttemptSelectRoot = "Select root file failed."

End If

End Function

Private Function AttemptSelectEF() As String

Dim nRet As Long

Dim szTemp As String

' Select an EF file

szTemp = Chr\$(&H3D) + Chr\$(&H40)

nRet = SelectEF(szTemp)

If nRet <> 0 Then

' Error - failed to select EF file

AttemptSelectEF = "Select EF file failed."

End If

End Function

Private Function AttemptVerify() As String

Dim nRet As Long

' Send the verify key

Dim VerifyKey As String

VerifyKey = Chr\$(1) + Chr\$(2) + Chr\$(3) + Chr\$(4) + Chr\$(5) + Chr\$(6) + Chr\$(7) + Chr\$(8)

modCardInterface - 2

```
nRet = SendVerifyKeyToEF(VerifyKey)
If nRet Then
    AttemptVerify = "Verify failed."
End If
```

Function

```
Public Function OpenReader(Optional Port As Long, Optional Baud As Long) As Long
    OpenReader = mvarReader.OpenReader
End Function
```

```
Public Function CardOn(Optional szKeyFileName As String) As Long
    Dim nRet As Long

    nRet = mvarReader.BootCard(szKeyFileName)

    ' Check the return val for success
    If (InStr(1, Hex$(nRet), "9000", vbTextCompare) = 0) And (InStr(1, Hex$(nRet), "9001", vbTextC
ompare) = 0) Then
        ' Failed to open the card.
        CardOn = True
    Else
        ' Success
        CardOn = False
    End If

End Function
```

```
Public Function SelectRootFile(szRoot As String) As Long
```

```
    Dim nRet As Long

    ' Pass the szRoot value as a string
    nRet = mvarReader.SelectDF(2, szRoot)

    ' Check the return val for success
    If (InStr(1, Hex$(nRet), "61", vbTextCompare) = 0) Then
        ' Failed to open the card.
        SelectRootFile = True
    Else
        ' Success
        SelectRootFile = False
    End If
```

End Function

```
Public Function SelectEF(szEF_File As String) As Long
```

```
    Dim nRet As Long

    ' Pass the nFile value as a string
    nRet = mvarReader.SelectEF(2, szEF_File)

    ' Check the return val for success
    If (InStr(1, Hex$(nRet), "61", vbTextCompare) = 0) Then
        ' Failed to open the card.
        SelectEF = True
    Else
        ' Success
        SelectEF = False
    End If
```

End Function

```
Public Function SendVerifyKeyToEF(szVerifyKey As String) As Long
```

```
    Dim nRet As Long
    ' Present the verify key to the smart card
    nRet = mvarReader.PresentVerify(Len(szVerifyKey), szVerifyKey)
```


modCardInterface - 3

```
' Check the return val for success
If (InStr(1, Hex$(nRet), "9000", vbTextCompare) = 0) And (InStr(1, Hex$(nRet), "9001", vbTextC
ompare) = 0) Then
    ' Failed to open the card.
    SendVerifyKeyToEF = True
Else
    ' Success
    SendVerifyKeyToEF = False
End If
```

End Function

Public Function CloseReader() As Long

```
' The return value is not really important here, but return it anyway.
CloseReader = mvarReader.CloseReader
```

End Function

Public Function ReadBinary(nOffset As Long, nDataLen As Long, szBuffer As String) As Long

' Do a binary read on the card

Dim nRet As Long

' Adjust the byte width for the card

If nOffset > 0 Then

nOffset = Int(nOffset \ DATA_WIDTH)

End If

' Size the buffer with room to spare, and clear it.

szBuffer = Space\$(255)

' Read the data from the card

If mvarReader Is Nothing Then MsgBox "problem, chief."

nRet = mvarReader.ReadCard(nOffset, nDataLen, szBuffer)

' Trim the buffer to the requested data length. Including spaces.

szBuffer = Left\$(szBuffer, nDataLen)

If InStr(1, Hex\$(nRet), "90", vbTextCompare) > 0 Then

' Read succeeded

ReadBinary = 0

Else

' Return the failure code.

ReadBinary = nRet

End If

End Function

Public Function WriteBinary(nOffset As Long, nDataLen As Long, szBuffer As String) As Long

Dim nRet As Long

' Adjust the byte width for the card

If nOffset > 0 Then

nOffset = Int(nOffset \ DATA_WIDTH)

End If

WriteBinary = mvarReader.WriteCard(nOffset, nDataLen, szBuffer)

End Function

Public Function GetMasterIDString() As String

' Returns the Master ID string from the first offset position on the card.

Dim nRet As Long

Dim szMasterID As String

Dim szBuffer As String * 255

' The master ID string will always be in the same location by design.

nRet = ReadBinary(0, 4, szBuffer)

If nRet = 0 Then

' Read succeeded. Get the ID bytes

GetMasterIDString = Left\$(szBuffer, 4)

```

Else
    ' Fail - return empty string
    GetMasterIDString = ""
End If

```

```

End Function

```

```

Public Function GetFileStruct() As Long
' Returns the file structure type number

```

```

    Dim szID As String
    ' Get the master ID string
    szID = GetMasterIDString

    If szID <> "" Then
        ' Convert it to a literal number and return
        GetFileStruct = Asc(Mid$(szID, 4, 1))
    End If

```

```

End Function

```

```

Public Function GetPassword() As String
' Returns the password currently on a card.
' Assumes card has been booted and is good to go.

```

```

    Dim nDataLen As Long
    Dim nOffset As Long
    Dim szBuffer As String * 255
    Dim nRet As Long
    Dim nFileStruct As Long
    ' Find out how the current card is formatted.
    nFileStruct = GetFileStruct
    If nFileStruct Then
        Dim Card As New CSmartCard
        Dim field As New CField
        Set field = Card.Structure.Fields("UserPassword")
        nOffset = field.OffsetBytes
        nDataLen = field.Length
    Else

```

```

        ' Unrecognized file type
        GetPassword = ""
        Exit Function
    End If
    ' Read the password field for the given file type.
    nRet = ReadBinary(nOffset, nDataLen, szBuffer)
    GetPassword = UnPad(field, szBuffer)
    Set Card = Nothing

```

```

End Function

```

```

Public Function GetUserName() As String
' Returns the user name from the card.

```

```

    Dim nDataLen As Long
    Dim nOffset As Long
    Dim szBuffer As String * 255
    Dim nRet As Long
    Dim nFileStruct As Long

    ' Find out how the current card is formatted.
    nFileStruct = GetFileStruct
    If nFileStruct Then
        Dim Card As New CSmartCard
        Dim field As New CField
        Set field = Card.Structure.Fields("UserName")
        nOffset = field.OffsetBytes
        nDataLen = field.Length
    Else

```

modCardInterface - 5

```
' Unrecognized file type
GetUserName = ""
Exit Function
End If
```

```
Read the user name field for the given file type.
nRet = ReadBinary(nOffset, nDataLen, szBuffer)
' Extract it and return it.
GetUserName = UnPad(field, szBuffer)
```

End Function

```
Public Function GetKeyCount() As Long
' Returns the number of keys on the card
```

```
Dim szID As String
szID = GetMasterIDString
If szID <> "" Then
    GetKeyCount = Asc(Mid$(szID, 3, 1))
End If
```

End Function

```
Public Function GetCardType() As String
' Returns the type ID byte from the first offset position on the card.
```

```
Dim szType As String
Dim szCardTypes As String
Dim szID As String

! For reference
szCardTypes = CARD_TYPES
' Get the master ID bytes from the card
szID = GetMasterIDString
If szID <> "" Then
    ' Read succeeded. Get the type byte
    szType = Left$(szID, 1)
    If InStr(1, szCardTypes, szType, vbTextCompare) > 0 Then
        ' This is a recognized type. Return the type.
        GetCardType = szType
    Else
        ' This is a blank, or unknown type. Return blank type.
        GetCardType = TYPE_BLANK
    End If
End If

End If
```

End Function

```
Public Function GetAllowedKeyCount() As Long
' Returns the number of keys on this card that the card is allowed to have.
```

```
Dim szID As String
' Get the master ID string
szID = GetMasterIDString

If szID <> "" Then
    ' Convert it to a literal number and return
    GetAllowedKeyCount = Asc(Mid$(szID, 3, 1))
End If
```

End Function

```
Public Function IsMasterCard() As Boolean
' Returns true if the current card is a Master card.
Dim szType As String
```

```
szType = GetCardType
If szType = TYPE_MASTER Then
    IsMasterCard = True
```

modCardInterface - 6

```
Else
    IsMasterCard = False
End If
```

End Function

Public Function IsMasterLocked() As Boolean

' Returns true if the Mastercard is locked.

Dim szLockByte As String

' Get the Master ID from the card

szLockByte = GetMasterIDString

' Separate out the locking id byte

' 1 = locked. Anything else is unlocked, or undertermined as of yet.

If szLockByte <> "" Then

 If Asc(Mid\$(szLockByte, 2, 1)) = 1 Then

 IsMasterLocked = True

 Else

 IsMasterLocked = False

 End If

End If

End Function

Public Function IsNewMaster() As Boolean

' Returns true if the UserName is uninitialized on the card.

Dim szUserName As String

' Get the user name from the card

szUserName = GetUserName

' If the user name is blank, then this is a new master.

If (szUserName = "") And IsMasterCard Then

 IsNewMaster = True

Else

 IsNewMaster = False

End If

End Function

Public Function UnPad(field As CField, szBuffer As String) As String

' Takes a field description and a buffer and returns the data

' extracted from the field, minus the padding character.

' Separate the username from its padding characters.

Dim p As Long

Dim n As Long

' See if the field has padding

p = InStr(1, szBuffer, field.PadChar, vbBinaryCompare)

If p = 0 Then

 ' There is no padding in this field, which will only occur when

 ' the field is full. Fix p to the len of the data.

 p = field.Length

 n = 0

Else

 ' Otherwise adjust for one-off

 n = 1

End If

UnPad = Trim\$(Left\$(szBuffer, p - n))

End Function

Public Function IsCardInserted() As Boolean

IsCardInserted = mvarReader.SenseCard

End Function

modMain - 1

Option Explicit

Private mvarMyForm As frmMain

Global Const CODEBOOK = "CodeBook"

Public mvarReader As CReaderEngine

Public mvarSmartCard As CSmartCard

Private Sub Main()

Dim nRet As Long

' Forbid multiple instantancing

If App.PreviousInstance Then End

frmSplash.Show

If mvarReader Is Nothing Then

' Create the card reader engine

Set mvarReader = CreateObject("ReaderEngine.CReaderEngine")

' Turn it on.

nRet = OpenReader

If nRet <> 0 Then

' Error occurred during initialization of the reader

MsgBox "Card reader failed to initialize." & "Error # " & "0x" & Hex\$(nRet)

' No point in continuing, can't talk to the card.

' End

End If

frmMain.Show

End If

End Sub

Public Sub ExitAdmin()

' Cleanup

' Dump the reader

mvarReader.CloseReader

' Release the resources

If Not mvarReader Is Nothing Then

Set mvarReader = Nothing

End If

If Not mvarSmartCard Is Nothing Then

Set mvarSmartCard = Nothing

End If

' Dump all the forms

End

End Sub

Public Function WriteKeys(nNextOffset As Long, Card As CSmartCard, keys As Collection) As Long

' Write the keys collection to the card. nNextOffset contains the

' byte offset at which the keys will be written.

Dim NewKey As CNewKey

Dim field As CField

For Each NewKey In keys

' Initialize a field with the default key values

Set field = Card.Copy("KeyID")

' Set the offset before beginning to calc the next offset

field.OffsetBytes = nNextOffset

nNextOffset = nNextOffset + field.Length

' Set the data in the field to the key name

field.Data = NewKey.KeyName

' Add the key under generic "KeyID", indexed by unique new keyname

Card.Fields.Add field, NewKey.KeyName

```

Set field = Card.Copy("KeyValue")
field.OffsetBytes = nNextOffset
nNextOffset = nNextOffset + field.Length
field.Data = NewKey.KeyValue
Card.Fields.Add field, NewKey.KeyValue

```

```

Set field = Card.Copy("KeyDate")
field.OffsetBytes = nNextOffset
nNextOffset = nNextOffset + field.Length
field.Data = NewKey.KeyDate
Card.Fields.Add field, CStr(NewKey.KeyDate) & NewKey.KeyName

```

```

Next
' Return the number of keys that were written, for no real reason.
WriteKeys = keys.Count

```

End Function

Public Function WriteHeader(Card As CSmartCard, Header As CCardHeader) As Long

```

Dim field As CField
' Create a copy of the desired field
' Add the data to the card's data collection

```

```

Set field = Card.Copy("MasterIDString")
field.Data = Header.MasterIDString
Card.Fields.Add field, field.Name

```

```

'KeyType , 1, -1, 0

```

```

Set field = Card.Copy("UserName")
field.Data = Header.UserName
Card.Fields.Add field, field.Name

```

```

Set field = Card.Copy("UserPassword")
field.Data = Header.UserPassword
Card.Fields.Add field, field.Name

```

```

Set field = Card.Copy("IntelligardID")
field.Data = Header.IntelligardID
Card.Fields.Add field, field.Name

```

```

Set field = Card.Copy("Counter")
field.Data = Header.TryCount
Card.Fields.Add field, field.Name

```

```

Set field = Card.Copy("Reserved")
field.Data = Header.Reserved
Card.Fields.Add field, field.Name

```

```

' Calculate the offset to the start of the key fields
WriteHeader = field.OffsetBytes + field.Length

```

End Function

Public Function GetNewID() As String

```

' Return either a random number, or the card's serial number
GetNewID = Trim$(Str$(Int((99999 * Rnd) + 10000)))

```

End Function

Public Sub DisplayKeys(lstKeys As ListBox, keys As Collection)

```

' Display the list of keys in the passed listbox

```

```

Dim key As CNewKey

```

```

' Clear any existing keys from the box.

```

```

lstKeys.Clear

```

```

' Load the box with the key names.

```

```

For Each key In keys

```

```

    lstKeys.AddItem key.KeyName

```

```

Next

```

End Sub

Public Function GetFileName(ByVal szName As String) As String

' Returns the file name without the path

Dim pStart As Long

Dim pEnd As Long

Dim a As String

Dim szSearchChar As String

szSearchChar = "\"

' Find the first path separator in the path

pStart = InStr(1, szName, szSearchChar)

If pStart = 0 Then

' No slashes found. Return the whole thing.

GetFileName = szName

Exit Function

End If

' If one was found, loop

Do While pStart > 0

pStart = InStr(1, szName, szSearchChar)

If pStart > 0 Then

' There was a sep char. Save all chars to the right of it.

szName = Right\$(szName, Len(szName) - pStart)

End If

' Loop for another path char.

pStart = InStr(1, szName, szSearchChar)

Loop

GetFileName = szName

End Function

Public Function GetFilePath(ByVal szFile As String)

' Returns just the path

Dim szTemp As String

szTemp = GetFileName(szFile)

GetFilePath = Left\$(szFile, Len(szFile) - Len(szTemp))

End Function

(No key file loaded)

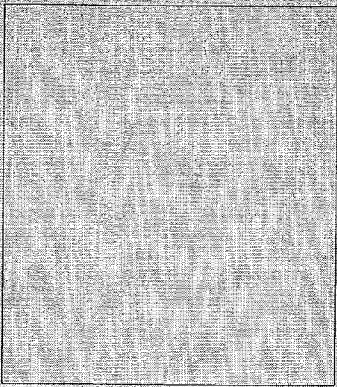


Key name:

[REDACTED]

[REDACTED]

[REDACTED]





User Name:

Password:




EXHIBIT Z-13

Smart Card Initialization

Console

Declaration of Stephen Zizzi



Transport Password
pinkvandthebrain

Encrypt Struct


☐ Smart Card
☒ Soft Card

☐ User
☒ Master

Number of Keys
10

File Struct
1

Verify Code (Hex)
FF FF FF FF FF FF FF FF

 Initialize

Command
1

Smart Card
Initialize Console

frmMain - 1

Option Explicit

Public info As New CInitInfo

Private Sub cmdFormat_Click()

' Shell to the DOS card formatting program.

MsgBox "Sorry, function not available."

End Sub

Private Sub cmdInitialize_Click()

Dim a As String

Dim x As Long

' Build the verify code string

For x = 0 To 7

a = a & Chr\$(txtDigit(x))

Next

With info

.ClearHeader = txtClearHeader

.Password = txtTransportPassword

.AllowedKeys = Val(txtNumKeys)

.FileStruct = Val(txtFileStruct)

.VerifyCode = a

If optMaster Then

.CardType = "M"

Else

.CardType = "B"

End If

' Call the appropriate data write function.

If Me.optSmartCard Then

InitSmartCard info

Else

InitSoftCard info

End If

End With

End Sub

Private Sub cmdMakeStruct_Click()

frmMain.cd.InitDir = App.Path

' Get the name of the file to encrypt

frmMain.cd.ShowOpen

End Sub

Private Sub Command1_Click()

ReadCardFile

End Sub

Private Sub Form_Load()

' Set the default transport password

Dim x As Long

For x = 0 To 7

txtDigit(x) = x + 1

Next

' Open the card reader

OpenReader

End Sub

Private Sub txtDigit_Change(Index As Integer)

frmMain - 2

```
Dim t As Long
' Set the char to uppercase for legibility
t = txtDigit(Index).SelStart
txtDigit(Index) = UCase$(txtDigit(Index))
txtDigit(Index).SelStart = t
```

End Sub

```
Private Sub txtDigit_GotFocus(Index As Integer)
txtDigit(Index).SelStart = 0
txtDigit(Index).SelLength = Len(txtDigit(Index))
End Sub
```

```
Private Sub txtDigit_KeyPress(Index As Integer, KeyAscii As Integer)
' Make sure it's a hex digit being entered.
If InStr(1, Chr$(8) & "0123456789ABCDEF", Chr$(KeyAscii), vbTextCompare) = 0 Then
KeyAscii = 0
End If
```

End Sub

modCardHandler -- 1

Option Explicit

Private Const STATUS_UNLOCKED = 1267609581#

Private Const STATUS_LOCKED = 1578145975#

Public Function InitSmartCard(info As CInitInfo) As Long

' Write the initial information to a smart card. At this

' point it should have been initialized with the DF and EF

' Flat file structure, which also puts in the verify password.

Dim szMID As String

' Write the master ID string

szMID = (info.CardType & Chr\$(0) & Chr\$(info.AllowedKeys) & Chr\$(info.FileStruct))

If IsCardInserted Then

WriteBinary 0, 4, szMID

End If

' Write the UserPassword

If IsCardInserted Then

Dim field As New CField

' Create a file structure for reference

Dim s As New CSmartCardFileStruct

s.Load info.FileStruct

' Find out where the user password is supposed to be

Set field = s.Fields("UserPassword")

field.Data = info.Password

info.Password = Pad(field)

' Write the password out

WriteBinary field.OffsetBytes, field.Length, info.Password

End If

End Function

Private Function Pad(field As CField) As String

' Pad the data field with the proper number of spaces.

Dim FieldLen As Long

Dim LenData As Long

Dim p As String

' Get the length the field should be

FieldLen = field.Length

' And the actual length right now

LenData = Len(field.Data)

' Create n pad characters in p

p = String(FieldLen - LenData, field.PadChar)

' Concatenate the padding to the end of the data and return.

Pad = field.Data & p

End Function

Public Function InitSoftCard(info As CInitInfo) As Long

Dim CardFile As CCardFile

Dim DF As New CDirectoryFile

Dim ef As New CElementaryFile

' Create a cardfile object

Set CardFile = New CCardFile

' Set up the clear header

Dim ClearHeader As CCLcarHeader

Set ClearHeader = CardFile.ClearHeader

ClearHeader.ClearText = info.ClearHeader

' Set up the encrypted header

Dim EncHeader As CMAZHeader

Set EncHeader = CardFile.EncryptedHeader

EncHeader.DataWidth = 4

EncHeader.DateCreated = Date

EncHeader.Expired = False

EncHeader.SerialNumber = 1965

EncHeader.Status = STATUS_UNLOCKED

```
Dim RootCollection As Collection
Set RootCollection = CardFile.RootDir
```

```
ef.EFKey = Chr$(&H3D) & Chr$(&H40)
Set EFData ef, info
Add the ef to the collection
DF.EFFileList.Add ef, ef.EFKey
' Add the df to the collection
DF.DFKey = Chr$(&H3D) & Chr$(&H0)
RootCollection.Add DF, DF.DFKey
```

```
frmMain.cd.InitDir = App.Path
```

```
' Get the name of the file to save the data under
frmMain.cd.ShowSave
```

```
' Open the file
```

```
Open frmMain.cd.FileName For Binary As 1
```

```
CardFile.Encrypt = True
```

```
' Write the card file
```

```
CardFile.WriteCardFile 1
```

```
Close 1
```

```
' Return success
```

```
InitSoftCard = 0
```

```
End Function
```

```
Private Function SetEFData(Efile As CElementaryFile, info As CInitInfo)
```

```
' Write the data into the expected locations in the EF
```

```
Dim field As New CField
```

```
' We know that the first four bytes will never change.
```

```
With Efile
```

```
.VerifyCode = info.VerifyCode
```

```
.SelStart = 0
```

```
.SelLength = 4
```

```
.SelData = (info.CardType & Chr$(0) & Chr$(info.AllowedKeys) & Chr$(info.FileStruct))
```

```
' Create a file structure for reference
```

```
Dim s As New CSmartCardFileStruct
```

```
s.Load info.FileStruct
```

```
' Find out where the user password is supposed to be
```

```
Set field = s.Fields("UserPassword")
```

```
' Write the password out
```

```
.SelStart = field.OffsetBytes
```

```
.SelLength = field.Length
```

```
.SelData = info.Password
```

```
End With
```

```
End Function
```

```
Public Function ReadCardFile()
```

```
Dim a As String
```

```
Dim nHeaderLen As Integer
```

```
Dim szHeader As String
```

```
Dim nOffset As Integer
```

```
Dim szCardData As String
```

```
Dim nClassID As Long
```

```
Dim EW As New CEncryptionServices
```

```
Dim CardFile As New CCardFile
```

```
frmMain.cd.InitDir = App.Path
```

```
' Get the name of the file to save the data under
```

```
frmMain.cd.ShowOpen
```


modCardHandler - 3

szCardData = frmMain.cd.FileName

' Create a clear header object
Dim ClearHeader As CClearHeader
Set ClearHeader = CardFile.ClearHeader

' Create an encrypted header object
Dim EncryptedHeader As CMAZHeader
Set EncryptedHeader = CardFile.EncryptedHeader

'Set mvarDirectoryFiles = CardFile.RootDir

Open szCardData For Binary Shared As 1

CardFile.Encrypt = True

CardFile.ReadCardFile 1

' Save the name of the file currently loaded, for later writing.

'mvarCurrentDataFile = szCardData

Close 1

End Function

CcCardFile - 1

Option Explicit

' This constant is also defined as 86 in CMAZHeader1

Private Const ERR_CORRUPT_CARDFILE = 86

Private mvarClearHeader As CClearHeader

Private mvarEncHeader As CMAZHeader

Private mvarRootDir As Collection

Private mvarEncrypt As Boolean

Private Const STATUS_UNLOCKED = 1267609581#

Private Const STATUS_LOCKED = 1578145975#

Private Const SMCODE = "fyd051mtld"

Private Sub Class_Initialize()

Set mvarRootDir = New Collection

Set mvarClearHeader = New CClearHeader1

Set mvarEncHeader = New CMAZHeader1

End Sub

Private Sub Class_Terminate()

Set mvarRootDir = Nothing

Set mvarEncHeader = Nothing

Set mvarClearHeader = Nothing

End Sub

Public Property Get Encrypt() As Boolean

Encrypt = mvarEncrypt

End Property

Public Property Let Encrypt(ByVal vData As Boolean)

mvarEncrypt = vData

End Property

Public Property Get DFCount() As Long

' Return the number of directory files in the root collection

DFCount = mvarRootDir.Count

End Property

Public Property Get ClearHeader() As CClearHeader

Set ClearHeader = mvarClearHeader

End Property

Public Property Get EncryptedHeader() As CMAZHeader

Set EncryptedHeader = mvarEncHeader

End Property

Public Property Get RootDir() As Collection

Set RootDir = mvarRootDir

End Property

Public Function WriteCardFile(ByVal nFile As Long, Optional ByVal nOffset As Long) As Boolean

' Write this object out to the file passed in nFile

Dim DF As CDirectoryFile

Dim a As String

Dim nCount As Long

Dim nEncStart As Long

' If no filemarker is passed, start at the current location.

If nOffset = 0 Then nOffset = Seek(nFile)

' Write the ClearHeader

mvarClearHeader.WriteClearHeader nFile

' Memorize where the encrypted header starts

nEncStart = Seek(nFile)

' Write the encrypted header

mvarEncHeader.WriteHeader nFile

' Write the number of DFs in the root dir

nCount = mvarRootDir.Count

Put #nFile, , nCount

' Now write the data structures

For Each DF In mvarRootDir

DF.WriteDF nFile

```

Next
If mvarEncrypt Then
    ' Now encrypt the part of the file that should be locked.
    Dim crack As New CEncryptionServices
    a = String(LOF(nFile) - nEncStart, Chr$(0))
    Get #nFile, nEncStart, a
    a = crack.Encrypt(a, SMCODE)
    Put #nFile, nEncStart, a
End If

```

End Function

Public Function ReadCardFile(ByVal nFile As Long, Optional ByVal nOffset As Long) As Long

' Write this object out to the file passed in nFile

Dim DF As CDirectoryFile

Dim a As String

Dim nCount As Long

Dim x As Long

Dim nEncStart As Long

If nOffset = 0 Then nOffset = Seek(nFile)

' Get the ClearHeader

mvarClearHeader.ReadClearHeader nFile

' Memorize where the encrypted header starts

nEncStart = Seek(nFile)

If mvarEncrypt Then

' Now decrypt the part of the file that is locked.

Dim crack As New CEncryptionServices

a = String(LOF(nFile) - nEncStart, Chr\$(0))

Get #nFile, nEncStart, a

a = crack.Decrypt(a, SMCODE)

Put #nFile, nEncStart, a

' Put the file marker back where it was

Seek #nFile, nEncStart

End If

' Get the encrypted header

If (mvarEncHeader.ReadHeader(nFile) = ERR_CORRUPT_CARDFILE) Then

ReadCardFile = ERR_CORRUPT_CARDFILE

Exit Function

End If

' Read the number of DFs in the root collection

Get #nFile, , nCount

' Now read the data structures

For x = 1 To nCount

Set DF = New CDirectoryFile

DF.ReadDF nFile

mvarRootDir.Add DF, DF.DFKey

Next

' Check if the card is supposed to be locked.

If mvarEncHeader.Status = STATUS_LOCKED Then

' The card is locked.

' Load an empty card to keep the reader from freaking out.

Set DF = New CDirectoryFile

Dim ef As New CElementaryFile

DF.EFileList.Add ef, "LockedEF"

mvarRootDir.Add DF, "LockedDF"

End If

If mvarEncrypt Then

a = String(LOF(nFile) - nEncStart, Chr\$(0))

Get #nFile, nEncStart, a

a = crack.Encrypt(a, SMCODE)

Put #nFile, nEncStart, a

End If

CCardFile - 3

End Function



CDirectoryFile - 1

Option Explicit

```
Private mvarEFileList As Collection
Private mvarDFKey As String
```

```
Private Sub Class_Initialize()
    Set mvarEFileList = New Collection
End Sub
```

```
Private Sub Class_Terminate()
    Set mvarEFileList = Nothing
End Sub
```

```
Public Property Set EFileList(ByVal vData As Object)
    Set mvarEFileList = vData
End Property
```

```
Public Property Get EFileList() As Collection
    Set EFileList = mvarEFileList
End Property
```

```
Public Property Get EFileCount() As Long
    EFileCount = mvarEFileList.Count
End Property
Public Property Let DFKey(ByVal vData As String)
    Let mvarDFKey = vData
End Property
```

```
Public Property Get DFKey() As String
    DFKey = mvarDFKey
End Property
```

```
Public Sub WriteDF(ByVal nFile As Long, Optional ByVal nOffset As Long)
```

```
    Dim ef As CElementaryFile
```

```
    Dim nFileCount As Long
```

```
    Dim nLen As Long
```

```
    ' If no filemarker has been given, take the current position
    If nOffset = 0 Then nOffset = Seek(nFile)
```

```
    nLen = Len(mvarDFKey)
```

```
    Put #nFile, nOffset, nLen
```

```
    Put #nFile, , mvarDFKey
```

```
    nFileCount = mvarEFileList.Count
```

```
    ' Save the elementary file count
```

```
    Put #nFile, , nFileCount
```

```
    ' Write out the efile list
```

```
    For Each ef In mvarEFileList
```

```
        ef.WriteEF nFile
```

```
    Next
```

```
End Sub
```

```
Public Sub ReadDF(ByVal nFile As Long, Optional ByVal nOffset As Long)
```

```
    ' Read in the efile list
```

```
    Dim nFileCount As Long
```

```
    Dim ef As CElementaryFile
```

```
    Dim x As Long
```

```
    Dim nLen As Long
```

```
    ' If no filemarker has been given, take the current position
```

```
    If nOffset = 0 Then nOffset = Seek(nFile)
```

```
    Get #nFile, nOffset, nLen
```

```
    mvarDFKey = String(nLen, Chr$(0))
```

```
    Get #nFile, , mvarDFKey
```

```
    ' Get the elementary file count
```

```
    Get #nFile, , nFileCount
```

```
    ' Load the ef collection
```

CDirectoryFile - 2

```
For x = 1 To nFileCount
    Set ef = New CElementaryFile
    ef.ReadEF nFile
    mvarEFileList.Add ef, ef.EFKey
```

Next

End Sub

CElementaryFile - 1

Option Explicit

```
Private mvarVerifyCode As String * 8
Private mvarEFKey As String
Private mvarFileSize As Long
```

```
Private mvarSelLength As Long
Private mvarSelStart As Long
Private mvarSelData As String
```

```
Private Sub Class_Initialize()
    ' Clear the buffer to null chars
    mvarSelData = String(4096, Chr$(0))
    mvarFileSize = Len(mvarSelData)
End Sub
```

```
Public Property Get FileSize() As Long
    FileSize = mvarFileSize
End Property
```

```
Public Property Get EFKey() As String
    EFKey = mvarEFKey
End Property
```

```
Public Property Let EFKey(vData As String)
    mvarEFKey = vData
End Property
```

```
Public Property Let SelData(ByVal vData As String)

    ' Set the data to whatever - adjust for one-off
    Dim SelStart As Long
    SelStart = mvarSelStart + 1

    If mvarSelLength = 0 Then
        ' Return the full string
        mvarSelData = vData
    Else
        Mid$(mvarSelData, SelStart, mvarSelLength) = vData
    End If

    ' Reset the selection start for the next data write
    mvarSelStart = 0
    mvarSelLength = 0
End Property
```

End Property

```
Public Property Get SelData() As String

    ' Set the data to whatever
    Dim SelStart As Long
    SelStart = mvarSelStart + 1
    If mvarSelLength = 0 Then
        ' Return the full string
        SelData = mvarSelData
    Else
        SelData = Mid$(mvarSelData, SelStart, mvarSelLength)
    End If

    ' Reset the selection start for the next data read
    SelStart = 0
    mvarSelLength = 0
End Property
```

End Property

```
Public Property Let SelStart(ByVal vData As Long)
    mvarSelStart = vData
End Property
```

```
Public Property Get SelStart() As Long
    SelStart = mvarSelStart
End Property
```

CElementaryFile - 2

End Property

```
Public Property Let SelLength(ByVal vData As Long)
    mvarSelLength = vData
End Property
```

```
Public Property Get SelLength() As Long
    SelLength = mvarSelLength
End Property
```

Public Sub Clear()

```
    ' Clear the buffer to null chars
    mvarSelData = String(4096, Chr$(0))
    mvarSelStart = 0
    mvarSelLength = 0
```

End Sub

```
Friend Property Let VerifyCode(Code As String)
    mvarVerifyCode = Code
End Property
```

```
Friend Property Get VerifyCode() As String
    VerifyCode = mvarVerifyCode
End Property
```

Public Function WriteEF(nFile As Long, Optional nOffset As Long) As Boolean

```
    ' Writes the contents of this object to nFile starting at nOffset
    Dim nLen As Long
```

```
    ' If no location is specified, start writing at the next available position.
    If nOffset = 0 Then nOffset = Seek(nFile)
```

```
    nLen = Len(mvarEFKey)
    Put #nFile, nOffset, nLen
    Put #nFile, , mvarEFKey
```

```
    nLen = Len(mvarVerifyCode)
    Put #nFile, , nLen
    Put #nFile, , mvarVerifyCode
```

```
    mvarFileSize = Len(SelData)
    Put #nFile, , mvarFileSize
    Put #nFile, , mvarSelData
```

End Function

Public Function ReadEF(nFile As Long, Optional nOffset As Long)

```
    ' Reads this object from nFile starting at nOffset
    Dim nLen As Long
```

```
    ' If no location is specified, start reading at the next available position.
    If nOffset = 0 Then nOffset = Seek(nFile)
```

```
    Get #nFile, nOffset, nLen
    mvarEFKey = String(nLen, Chr$(0))
    Get #nFile, , mvarEFKey
```

```
    Get #nFile, , nLen
    mvarVerifyCode = String(nLen, Chr$(0))
    Get #nFile, , mvarVerifyCode
```

```
    Get #nFile, , mvarFileSize
    'mvarFileSize = 4096
    mvarSelData = String(mvarFileSize, Chr$(0))
    Get #nFile, , mvarSelData
```

End Function

CField - 1

Option Explicit

```
Private mvarName As String
Private mvarLength As Long
Private mvarOffsetBytes As Long
Private mvarKeyType As Long
Private mvarPadChar As Byte
Private mvarPad As Boolean
Private mvarData As String
```

```
Public Property Let KeyType(ByVal vData As Long)
    mvarKeyType = vData
End Property
```

```
Public Property Get KeyType() As Long
    KeyType = mvarKeyType
End Property
Public Property Let Data(ByVal vData As String)
    mvarData = vData
End Property
```

```
Public Property Get Data() As String
    Data = mvarData
End Property
```

```
Friend Property Let Pad(ByVal vData As Boolean)
    mvarPad = vData
End Property
```

```
Friend Property Get Pad() As Boolean
    Pad = mvarPad
End Property
```

```
Friend Property Let PadChar(ByVal vData As String)
    mvarPadChar = Asc(vData)
End Property
```

```
Friend Property Get PadChar() As String
    PadChar = Chr$(mvarPadChar)
End Property
```

```
Friend Property Let OffsetBytes(ByVal vData As Long)
    mvarOffsetBytes = vData
End Property
```

```
Friend Property Get OffsetBytes() As Long
    OffsetBytes = mvarOffsetBytes
End Property
```

```
Friend Property Let Length(ByVal vData As Long)
    mvarLength = vData
End Property
```

```
Friend Property Get Length() As Long
    Length = mvarLength
End Property
```

```
Public Property Let Name(ByVal vData As String)
    mvarName = vData
End Property
```

```
Public Property Get Name() As String
    Name = mvarName
End Property
```

CInitInfo - 1

Option Explicit

Private mvarClearHeader As String

Private mvarPassword As String

Private mvarCardType As String

Private mvarAllowedKeys As Long

Private mvarFileStruct As Long

Private mvarVerifyCode As String

Public Property Let ClearHeader(ByVal vData As String)

mvarClearHeader = vData

End Property

Public Property Get ClearHeader() As String

ClearHeader = mvarClearHeader

End Property

Public Property Let FileStruct(ByVal vData As Long)

mvarFileStruct = vData

End Property

Public Property Get FileStruct() As Long

FileStruct = mvarFileStruct

End Property

Public Property Let AllowedKeys(ByVal vData As Long)

mvarAllowedKeys = vData

End Property

Public Property Get AllowedKeys() As Long

AllowedKeys = mvarAllowedKeys

End Property

Public Property Let CardType(ByVal vData As String)

mvarCardType = vData

End Property

Public Property Get CardType() As String

CardType = mvarCardType

End Property

Public Property Let Password(ByVal vData As String)

mvarPassword = vData

End Property

Public Property Get Password() As String

Password = mvarPassword

End Property

Public Property Let VerifyCode(ByVal vData As String)

mvarVerifyCode = vData

End Property

Public Property Get VerifyCode() As String

VerifyCode = mvarVerifyCode

End Property

CSmartCardFileStruct - 1

Option Explicit

Const SEP = "-"

Const mvar_def_ResourceFile = "Struct.mcf"

```
Private mvarStructID As Long      ' Holds the currently loaded structure ID
Private mvarFields As Collection  ' Holds the fields collection in the structure
Private mvarResourceFile As String ' Holds the name of the file containing the structure definitions
Private Const STRUCT_KEY = ".@.COxn3k."
Private Sub Class_Initialize()
    ' Set the default resource file for structure data
    mvarResourceFile = App.Path & "\" & mvar_def_ResourceFile
    ' Instantiate the fields collection
    Set mvarFields = New Collection
End Sub
```

End Sub

```
Public Function Load(nStruct As Long) As Boolean
    ' Load the requested file structure from an external resource
    Dim crack As New CEncryptionServices
    Dim a As String, b As String
```

```
    ' Check for the existence of the options file.
    If Dir(mvarResourceFile) = "" Then
        ' The structure file does not exist.  Oops.
        Load = False
        Exit Function
    Else
```

```
        ' Read in the options file.
        Open mvarResourceFile For Binary As 1
        ' Set up a buffer request for the whole file
        a = String(LOF(1), Chr$(0))
        ' Suck it in, memorize it
        Get #1, 1, a: b = a
        ' Decrypt it
        a = crack.Decrypt(a, STRUCT_KEY)
        ' Put it back clear
        Put #1, 1, a
        Close 1
    End If
```

```
Dim szStructName As String
Dim field As CField      ' Holds an individual field definition
Dim bError As Boolean    ' Error flag during load
Dim nTotalLength As Long ' For calculating file offsets
Dim Length As String     ' Generic for calculations
```

```
    ' Set the structure's loaded structure ID
    mvarStructID = nStruct
    If nStruct > 0 Then
        ' Convert the file structure number into a string
        szStructName = Trim$(Str$(nStruct))
        ' Open the file structure definitions
        Open mvarResourceFile For Input As 1
        ' Find the structure identifier in the file.
        If SetFilePointer(szStructName) Then
            ' Couldn't find the structure definition header.
            bError = True
        Else
            ' Found the structure definition. Load it in.
            Do While Not EOF(1)
                ' Get a field from the file
                If GetNextField(a) Then
                    ' Got a field.
                    ' Check to make sure that wasn't the end of the definition.
                End If
            Loop
        End If
    End If
```

```

    If UCase$(a) = "-END-" Then
        ' If it was, cut out.
        Exit Do
    Else
        ' Instantiate a new field
        Set field = ParseDefinition(a)
        ' Calculate the byte offset
        field.OffsetBytes = nTotalLength
        Length = field.Length
        nTotalLength = nTotalLength + Length

        ' Add the field to the collection
        mvarFields.Add field, field.Name
    End If
End If

' Keep going until we hit the end of the file, or the end statement.
Loop

```

```

    End If
    Close 1
Else
    ' No such file struct
    bError = True
End If

' Return the status of the load. Success, or failure.
Load = bError

' Release the resources.
Set field = Nothing

```

```

-----
' Re-open for sequential access for encryption
Open mvarResourceFile For Binary As 1
' Replace the encrypted text
Put #1, 1, b
Close 1

```

End Function

Private Function SetFilePointer(szName As String) As Boolean

' Parse a file for a given text string, leave the file pointer

' at the line after the header.

```

    Dim a As String
    Dim bFound As Boolean

    ' Note: File must be opened prior to this call
    Do While Not EOF(1)
        ' Parse for the required file structure
        Input #1, a
        If a = szName Then
            bFound = True
            Exit Do
        End If
    Loop

    ' Return the false if we found it.
    If bFound Then
        SetFilePointer = False
    Else
        SetFilePointer = True
    End If

```

End Function

Private Function ParseDefinition(szDef As String) As CField

Dim field As New CField

Dim Name As String

CSmartCardFileStruct - 3

```
Dim Length As Long
Dim Pad As Boolean
Dim PadChar As String
Dim p As Long
Dim nTotalLength As Long
```

' Parse out each of the parameters

```
p = InStr(1, szDef, ",", vbTextCompare)
Name = Left$(szDef, p - 1)
szDef = Trim$(Right$(szDef, Len(szDef) - p))
```

```
p = InStr(1, szDef, ",", vbTextCompare)
Length = Val(Left$(szDef, p - 1))
szDef = Trim$(Right$(szDef, Len(szDef) - p))
```

```
p = InStr(1, szDef, ",", vbTextCompare)
Pad = Left$(szDef, p - 1)
szDef = Trim$(Right$(szDef, Len(szDef) - p))
```

```
PadChar = Chr$(szDef)
```

' Set the definitions to the field

With field

```
.Name = Name: .Length = Length: .Pad = Pad: .PadChar = PadChar
End With
```

Set ParseDefinition = field

End Function

Private Function GetNextField(szField As String)

' Parse the file until a valid field definition is found

' NOTE: This function assumes that the file pointer

' has been positioned to the line after the selected header.

```
Dim a As String
```

```
Dim bFound As Boolean
```

```
Do While Not EOF(1)
```

```
Line Input #1, a
```

' Make sure it's not a blank line or a comment

```
If (Left$(a, 1) <> ";") And (Trim$(a) <> "") Then
```

' Return a valid line

```
bFound = True
```

```
szField = a
```

```
Exit Do
```

```
End If
```

```
Loop
```

```
GetNextField = bFound
```

End Function

Friend Property Get Fields() As Collection

```
Set Fields = mvarFields
```

End Property

Friend Property Get StructID() As Long

```
StructID = mvarStructID
```

End Property

Friend Property Let StructFile(szNewFilename As String)

' Set the file name for the location of the file structures.


```
mvarResourceFile = szNewFilename
```

End Property

EXHIBIT Z-14

Smart Card Reader Engine

Declaration of Stephen Zizzi



modMain - 1

Smart Card

Reader

6/97

' Smart Card Reader Engine

' SJZ june 97

Option Explicit

Public Gank As CTimer

Public Poll As CTimer

Public Engine As CReaderEngine

Public Function GankProc(ByVal hwnd As Long, ByVal msg As Long, ByVal idEvent As Long, ByVal dwTime As Long) As Long

Gank.Enabled = False

' Should return 0. We dealt with it.

GankProc = 0

End Function

Public Function PollProc(ByVal hwnd As Long, ByVal msg As Long, ByVal idEvent As Long, ByVal dwTime As Long) As Long

Engine.PollReader

' Should return 0

PollProc = 0

End Function

CcCardFile - 1

Option Explicit

' This constant is also defined as 86 in CMAZHeader1

Private Const ERR_CORRUPT_CARDFILE = 86

Private mvarClearHeader As CClearHeader

Private mvarEncHeader As CMAZHeader

Private mvarRootDir As Collection

Private mvarEncrypt As Boolean

Private Const STATUS_UNLOCKED = 1267609581#

Private Const STATUS_LOCKED = 1578145975#

Private Const SMCODE = "fyd05lmtld"

Private Sub Class_Initialize()

Set mvarRootDir = New Collection

Set mvarClearHeader = New CClearHeader1

Set mvarEncHeader = New CMAZHeader1

End Sub

Private Sub Class_Terminate()

Set mvarRootDir = Nothing

Set mvarEncHeader = Nothing

Set mvarClearHeader = Nothing

End Sub

Public Property Get Encrypt() As Boolean

Encrypt = mvarEncrypt

End Property

Public Property Let Encrypt(ByVal vData As Boolean)

mvarEncrypt = vData

End Property

Public Property Get DFCount() As Long

' Return the number of directory files in the root collection

DFCount = mvarRootDir.Count

End Property

Public Property Get ClearHeader() As CClearHeader

Set ClearHeader = mvarClearHeader

End Property

Public Property Get EncryptedHeader() As CMAZHeader

Set EncryptedHeader = mvarEncHeader

End Property

Public Property Get RootDir() As Collection

Set RootDir = mvarRootDir

End Property

Public Function WriteCardFile(ByVal nFile As Long, Optional ByVal nOffset As Long) As Boolean

' Write this object out to the file passed in nFile

Dim DF As CDirectoryFile

Dim a As String

Dim nCount As Long

Dim nEncStart As Long

' If no filemarker is passed, start at the current location.

If nOffset = 0 Then nOffset = Seek(nFile)

' Write the ClearHeader

mvarClearHeader.WriteClearHeader nFile

' Memorize where the encrypted header starts

nEncStart = Seek(nFile)

' Write the encrypted header

mvarEncHeader.WriteHeader nFile

' Write the number of DFs in the root dir

nCount = mvarRootDir.Count

Put #nFile, , nCount

' Now write the data structures

For Each DF In mvarRootDir

DF.WriteDF nFile


```

Next
If mvarEncrypt Then
    ' Now encrypt the part of the file that should be locked.
    Dim crack As New CEncryptionServices
    a = String(LOF(nFile) - nEncStart, Chr$(0))
    Get #nFile, nEncStart, a
    a = crack.Encrypt(a, SMCODE)
    Put #nFile, nEncStart, a
End If

```

End Function

```

Public Function ReadCardFile(ByVal nFile As Long, Optional ByVal nOffset As Long) As Long
' Write this object out to the file passed in nFile

```

```

    Dim DF As CDirectoryFile
    Dim a As String
    Dim nCount As Long
    Dim x As Long
    Dim nEncStart As Long

```

```

    If nOffset = 0 Then nOffset = Seek(nFile)
    ' Get the ClearHeader
    mvarClearHeader.ReadClearHeader nFile

```

```

    ' Memorize where the encrypted header starts
    nEncStart = Seek(nFile)

```

```

    If mvarEncrypt Then
        ' Now decrypt the part of the file that is locked.
        Dim crack As New CEncryptionServices
        a = String(LOF(nFile) - nEncStart, Chr$(0))
        Get #nFile, nEncStart, a

```

```

        a = crack.Decrypt(a, SMCODE)
        Put #nFile, nEncStart, a
        ' Put the file marker back where it was
        Seek #nFile, nEncStart

```

End If

```

    ' Get the encrypted header
    If (mvarEncHeader.ReadHeader(nFile) = ERR_CORRUPT_CARDFILE) Then
        ReadCardFile = ERR_CORRUPT_CARDFILE
        Exit Function
    End If

```

```

    ' Read the number of DFs in the root collection
    Get #nFile, , nCount

```

```

    ' Now read the data structures
    For x = 1 To nCount
        Set DF = New CDirectoryFile
        DF.ReadDF nFile
        mvarRootDir.Add DF, DF.DFKey
    Next

```

```

    ' Check if the card is supposed to be locked.
    If mvarEncHeader.Status = STATUS_LOCKED Then
        ' The card is locked.
        ' Load an empty card to keep the reader from freaking out.
        Set DF = New CDirectoryFile
        Dim ef As New CElementaryFile
        DF.EFileList.Add ef, "LockedEF"
        mvarRootDir.Add DF, "LockedDF"
    End If

```

```

    If mvarEncrypt Then
        a = String(LOF(nFile) - nEncStart, Chr$(0))
        Get #nFile, nEncStart, a
        a = crack.Encrypt(a, SMCODE)
        Put #nFile, nEncStart, a
    End If

```

End Function



CCardInterface - 1

Option Explicit

' Virtual Hardware/Cardware interface

Public Function OpenReader() As Long

End Function

Public Function CloseReader() As Long

End Function

Public Function BootCard(Optional szBuffer As String) As Long

End Function

Public Function SenseCard() As Long

End Function

Public Function ReadBinary(nOffset As Long, nDataLen As Long, szBuffer As String) As Long

End Function

Public Function WriteBinary(nOffset As Long, nDataLen As Long, szData As String) As Long

End Function

Public Function PresentVerify(CodeLen As Long, Code As String) As Long

End Function

Public Function SelectEF(ByVal DataLen As Long, ByVal EFile As String) As Long

End Function

Public Function SelectDF(ByVal DataLen As Long, ByVal DFile As String) As Long

End Function

Public Function DataWidth(ByVal Width As Long) As Long

End Function

Public Function ForceLock() As Long

End Function

Public Function SerialNumber() As String

End Function

CDirectoryFile - 1

Option Explicit

Private mvarEFileList As Collection
Private mvarDFKey As String

Private Sub Class_Initialize()
Set mvarEFileList = New Collection
End Sub

Private Sub Class_Terminate()
Set mvarEFileList = Nothing
End Sub

Public Property Set EFileList(ByVal vData As Object)
Set mvarEFileList = vData
End Property

Public Property Get EFileList() As Collection
Set EFileList = mvarEFileList
End Property

Public Property Get EFileCount() As Long
EFileCount = mvarEFileList.Count
End Property
Public Property Let DFKey(ByVal vData As String)
Let mvarDFKey = vData
End Property

Public Property Get DFKey() As String
DFKey = mvarDFKey
End Property

Public Sub WriteDF(ByVal nFile As Long, Optional ByVal nOffset As Long)

Dim ef As CElementaryFile
Dim nFileCount As Long
Dim nLen As Long

' If no filemarker has been given, take the current position
If nOffset = 0 Then nOffset = Seek(nFile)

nLen = Len(mvarDFKey)
Put #nFile, nOffset, nLen
Put #nFile, , mvarDFKey
nFileCount = mvarEFileList.Count
' Save the elementary file count
Put #nFile, , nFileCount

' Write out the efile list
For Each ef In mvarEFileList
ef.WriteEF nFile
Next

End Sub

Public Sub ReadDF(ByVal nFile As Long, Optional ByVal nOffset As Long)

' Read in the efile list
Dim nFileCount As Long
Dim ef As CElementaryFile
Dim x As Long
Dim nLen As Long

' If no filemarker has been given, take the current position
If nOffset = 0 Then nOffset = Seek(nFile)
Get #nFile, nOffset, nLen
mvarDFKey = String(nLen, Chr\$(0))
Get #nFile, , mvarDFKey
' Get the elementary file count
Get #nFile, , nFileCount

' Load the ef collection

CDirectoryFile - 2

```
For x = 1 To nFileCount  
    Set ef = New CElementaryFile  
    ef.ReadEF nFile  
    mvarEFileList.Add ef, ef.EFKey
```

Next

End Sub

CElementaryFile - 1

Option Explicit

```
Private mvarVerifyCode As String * 8
Private mvarEFKey As String
Private mvarFileSize As Long
```

```
Private mvarSelLength As Long
Private mvarSelStart As Long
Private mvarSelData As String
```

```
Private Sub Class_Initialize()
    ' Clear the buffer to null chars
    mvarSelData = String(4096, Chr$(0))
    mvarFileSize = Len(mvarSelData)
End Sub
```

```
Public Property Get FileSize() As Long
    FileSize = mvarFileSize
End Property
```

```
Public Property Get EFKey() As String
    EFKey = mvarEFKey
End Property
```

```
Public Property Let EFKey(vData As String)
    mvarEFKey = vData
End Property
```

```
Public Property Let SelData(ByVal vData As String)
    ' Set the data to whatever - adjust for one-off
    Dim SelStart As Long
    SelStart = mvarSelStart + 1

    If mvarSelLength = 0 Then
        ' Return the full string
        mvarSelData = vData
    Else
        Mid$(mvarSelData, SelStart, mvarSelLength) = vData
    End If

    ' Reset the selection start for the next data write
    mvarSelStart = 0
    mvarSelLength = 0
End Property
```

End Property

```
Public Property Get SelData() As String
```

```
    ' Set the data to whatever
    Dim SelStart As Long
    SelStart = mvarSelStart + 1
    If mvarSelLength = 0 Then
        ' Return the full string
        SelData = mvarSelData
    Else
        SelData = Mid$(mvarSelData, SelStart, mvarSelLength)
    End If
    ' Reset the selection start for the next data read
    SelStart = 0
    mvarSelLength = 0
End Property
```

End Property

```
Public Property Let SelStart(ByVal vData As Long)
    mvarSelStart = vData
End Property
```

```
Public Property Get SelStart() As Long
    SelStart = mvarSelStart
```

End Property

Public Property Let SelLength(ByVal vData As Long)
 mvarSelLength = vData

End Property

Public Property Get SelLength() As Long
 SelLength = mvarSelLength

End Property

Public Sub Clear()

' Clear the buffer to null chars
 mvarSelData = String(4096, Chr\$(0))
 mvarSelStart = 0
 mvarSelLength = 0

End Sub

Friend Property Let VerifyCode(Code As String)
 mvarVerifyCode = Code

End Property

Friend Property Get VerifyCode() As String
 VerifyCode = mvarVerifyCode

End Property

Public Function WriteEF(nFile As Long, Optional nOffset As Long) As Boolean

' Writes the contents of this object to nFile starting at nOffset
 Dim nLen As Long

' If no location is specified, start writing at the next available position.
 If nOffset = 0 Then nOffset = Seek(nFile)

nLen = Len(mvarEFKey)
 Put #nFile, nOffset, nLen
 Put #nFile, , mvarEFKey

nLen = Len(mvarVerifyCode)
 Put #nFile, , nLen
 Put #nFile, , mvarVerifyCode

mvarFileSize = Len(SelData)
 Put #nFile, , mvarFileSize
 Put #nFile, , mvarSelData

End Function

Public Function ReadEF(nFile As Long, Optional nOffset As Long)

' Reads this object from nFile starting at nOffset
 Dim nLen As Long

' If no location is specified, start reading at the next available position.
 If nOffset = 0 Then nOffset = Seek(nFile)

Get #nFile, nOffset, nLen
 mvarEFKey = String(nLen, Chr\$(0))
 Get #nFile, , mvarEFKey

Get #nFile, , nLen
 mvarVerifyCode = String(nLen, Chr\$(0))
 Get #nFile, , mvarVerifyCode

Get #nFile, , mvarFileSize
 'mvarFileSize = 4096
 mvarSelData = String(mvarFileSize, Chr\$(0))
 Get #nFile, , mvarSelData

End Function

```
' Defines the GEMPLUS card hardware interface
```

```
Option Explicit
```

```
Implements CHardCard
```

```
' New function decs
```

```
Public Declare Function CLX_OpenReader Lib "screader.dll" (ByVal Port As Integer, ByVal Baud As Integer) As Long
```

```
Private Declare Function CLX_CloseReader Lib "screader.dll" () As Long
```

```
Private Declare Function CLX_CardOn Lib "screader.dll" (ByVal rdata As String) As Long
```

```
Private Declare Function MAZ_SelectDF Lib "screader.dll" (ByVal DataLen As Long, ByVal DataBuf As String) As Long
```

```
Private Declare Function MAZ_SelectEF Lib "screader.dll" (ByVal DataLen As Long, ByVal DataBuf As String) As Long
```

```
Private Declare Function CLX_Verify7816 Lib "screader.dll" (ByVal DataLen As Long, ByVal DataBuf As String) As Long
```

```
Private Declare Function CLX_CardInserted Lib "screader.dll" () As Boolean
```

```
Private Declare Function CLX_ReadBinary7816 Lib "screader.dll" (ByVal Address As Long, ByVal DataLen As Long, ByVal DataBuf As String) As Long
```

```
Private Declare Function CLX_WriteBinary7816 Lib "screader.dll" (ByVal Address As Long, ByVal DataLen As Long, ByVal DataBuf As String) As Long
```

```
' =====
```

```
Private Declare Function CLX_ResetReader Lib "screader.dll" () As Long
```

```
Private Declare Function CLX_GreenLedOn Lib "screader.dll" () As Long
```

```
Private Declare Function CLX_GreenLedOff Lib "screader.dll" () As Long
```

```
Private Declare Function CLX_RedLedOn Lib "screader.dll" () As Long
```

```
Private Declare Function CLX_RedLedOff Lib "screader.dll" () As Long
```

```
Private Declare Function CLX_GetReaderStatus Lib "screader.dll" () As Long
```

```
Private Declare Function CLX_SetCardType Lib "screader.dll" (ByVal x As Integer) As Long
```

```
Private Declare Function CLX_ReadCard Lib "screader.dll" (ByVal rdata As Any, ByVal addr As Long, ByVal Length As Integer) As Long
```

```
Private Declare Function CLX_WriteCard Lib "screader.dll" (ByVal rdata As Any, ByVal addr As Long, ByVal Length As Integer) As Long
```

```
Private Declare Function CLX_GetVersion Lib "screader.dll" (ByVal rdata As Any) As Long
```

```
' Async Processor Cards
```

```
Private Declare Function MAZ_SetCardStatus Lib "screader.dll" () As Long
```

```
Private Declare Function CLX_CardOff Lib "screader.dll" () As Long
```

```
Private Declare Function CLX_Invalidate7816 Lib "screader.dll" () As Long
```

```
Private Declare Function CLX_Rehab7816 Lib "screader.dll" () As Long
```

```
Private Declare Function CLX_UpdateBinary7816 Lib "screader.dll" (ByVal Address As Integer, ByVal P3 As Long, ByVal DataBuf As Any) As Long
```

```
Private Declare Function CLX_Select7816 Lib "screader.dll" (ByVal DataLen As Long, ByVal DataBuf As Any) As Long
```

```
Private Declare Function CLX_Change_Code7816 Lib "screader.dll" (ByVal DataLen As Long, ByVal DataBuf As Any) As Long
```

```
Private Function CHardCard_BootCard(Optional szBuffer As String) As Long
```

```
    If szBuffer = "" Then
```

```
        szBuffer = String(255, " ")
```

```
    End If
```

```
    CHardCard_BootCard = CLX_CardOn(szBuffer)
```

```
End Function
```

```
Private Function CHardCard_CardOff() As Long
```

```
End Function
```

```
Private Function CHardCard_ChangeCode(ByVal nDataLen As Long, szData As String) As Long
```

```
End Function
```

```
Private Function CHardCard_CloseReader() As Long
```

```
    CHardCard_CloseReader = CLX_CloseReader
```

```
End Function
```



```

Private Function CHardCard_DataWidth(ByVal nWidth As Long) As Long
    ' Set the data width on the card
    ' CLX_WriteBinary7816(nOffset, nDataLen, nWidth)
End Function

Private Function CHardCard_GetVersion(szData As String) As Long
End Function

Private Function CHardCard_Invalidate() As Long
    CHardCard_Invalidate = CLX_Invalidate7816
End Function

Private Function CHardCard_OpenReader(Optional ByVal nPort As Long, Optional ByVal nBaud As Long)
As Long
    ' Initialize the variables to zero
    nPort = nPort Or 0
    nBaud = nBaud Or 0
    Dim n As Long
    On Error GoTo duh
    n = CLX_OpenReader(nPort, nBaud)

    CHardCard_OpenReader = True
Exit Function
duh:
    ' Something went wrong during initialization.
    If Err.Number = 48 Then
        MsgBox "Unable to start IntelliGard. No hardware interface detected."
    Else
        MsgBox "Unable to start IntelliGard. You may need to reinstall the software."
    End If
    ' Kill the proc.
    CHardCard_OpenReader = False
End Function

Private Function CHardCard_ReadBinary(ByVal nOffset As Long, ByVal nDataLen As Long, szData As Str
ing) As Long
    CHardCard_ReadBinary = CLX_ReadBinary7816(nOffset, nDataLen, szData)
End Function

Private Function CHardCard_ReaderStatus() As Long
End Function

Private Function CHardCard_Rehab() As Long
End Function

Private Function CHardCard_Reset() As Long
End Function

Private Function CHardCard_SelectDF(ByVal nDataLen As Long, ByVal szData As String) As Long
    CHardCard_SelectDF = MAZ_SelectDF(nDataLen, szData)
End Function

Private Function CHardCard_SelectEF(ByVal nDataLen As Long, ByVal szData As String) As Long
    CHardCard_SelectEF = MAZ_SelectEF(nDataLen, szData)
End Function

Private Function CHardCard_SenseCard() As Long
    Static bReaderOn As Boolean
    Dim n As Long

    If bReaderOn Then
        ' The reader is on, which means the card was inserted last time. See if it still is.
        n = CLX_CardInserted
        If n = False Then

```

```
' The card is gone. With NT, the reader must now be expressly closed.
```

```
CHardCard_CloseReader
```

```
bReaderOn = False
```

```
End If
```

```
Else
```

```
' The reader is not on. Turn it on.
```

```
If CHardCard_OpenReader Then
```

```
' Check for the card.
```

```
n = CLX_CardInserted
```

```
If n Then
```

```
' As long as the card is inserted, the reader will remain on.
```

```
bReaderOn = True
```

```
Else
```

```
' No card in reader - prepare to cycle reader on/off repeatedly
```

```
bReaderOn = False
```

```
CHardCard_CloseReader
```

```
End If
```

```
Else
```

```
' Reader could not be opened.
```

```
n = False
```

```
End If
```

```
End If
```

```
' Pass back the result.
```

```
CHardCard_SenseCard = n
```

```
End Function
```

```
Private Function CHardCard_SerialNumber() As String
```

```
'CHardCard_SerialNumber = CLX whatever
```

```
End Function
```

```
Private Function CHardCard_SetCardStatus() As Long
```

```
End Function
```

```
Private Function CHardCard_SetCardType(ByVal nType As Integer) As Long
```

```
End Function
```

```
Private Function CHardCard_Verify(ByVal nDataLen As Long, ByVal szData As String) As Long
```

```
CHardCard_Verify = CLX_Verify7816(nDataLen, szData)
```

```
End Function
```

```
Private Function CHardCard_WriteBinary(ByVal nOffset As Long, ByVal nDataLen As Long, szData As String) As Long
```

```
CHardCard_WriteBinary = CLX_UpdateBinary7816(nOffset, nDataLen, szData)
```

```
End Function
```

CHardCard - 1

Option Explicit

' Virtual Hardware interface

Public Function OpenReader(Optional ByVal nPort As Long, Optional ByVal nBaud As Long) As Long

End Function

Public Function BootCard(Optional szBuffer As String) As Long

End Function

Public Function SelectDF(ByVal nDataLen As Long, ByVal szData As String) As Long

End Function

Public Function SelectEF(ByVal nDataLen As Long, ByVal szData As String) As Long

End Function

Public Function Verify(ByVal nDataLen As Long, ByVal szData As String) As Long

End Function

Public Function SenseCard() As Long

End Function

Public Function CloseReader() As Long

End Function

Public Function ReadBinary(ByVal nOffset As Long, ByVal nDataLen As Long, szData As String) As Long

End Function

Public Function WriteBinary(ByVal nOffset As Long, ByVal nDataLen As Long, szData As String) As Long

End Function

Public Function DataWidth(ByVal nWidth As Long) As Long

End Function

Public Function ChangeCode(ByVal nDataLen As Long, szData As String) As Long

End Function

Public Function Rehab() As Long

End Function

Public Function Invalidate() As Long

End Function

Public Function CardOff() As Long

End Function

Public Function SetCardStatus() As Long

End Function

Public Function GetVersion(szData As String) As Long

End Function

Public Function SetCardType(ByVal nType As Integer) As Long

End Function

Public Function ReaderStatus() As Long

End Function

Public Function SerialNumber() As String

End Function

'Public Property Let HRedLED(ByVal vData As Boolean)

'End Property

'Public Property Get HRedLED() As Boolean

'End Property

'Public Property Let HGreenLED(ByVal vData As Boolean)

'End Property

'Public Property Get HGreenLED() As Boolean

'End Property

Public Function Reset() As Long

End Function

CHardwareInterface - 1

Option Explicit

Implements CCardInterface

' Table of available device interfaces.

Private Const GEM_PLUS = 1

Private mvarDevice As CHardCard

Private Function CCardInterface_BootCard(Optional szBuffer As String) As Long

CCardInterface_BootCard = mvarDevice.BootCard

End Function

Private Function CCardInterface_CloseReader() As Long

CCardInterface_CloseReader = mvarDevice.CloseReader

End Function

Private Function CCardInterface_DataWidth(ByVal Width As Long) As Long

CCardInterface_DataWidth = mvarDevice.DataWidth(Width)

End Function

Private Function CCardInterface_ForceLock() As Long

CCardInterface_ForceLock = mvarDevice.Invalidate

End Function

Private Function CCardInterface_OpenReader() As Long

CCardInterface_OpenReader = mvarDevice.OpenReader

End Function

Private Function CCardInterface_PresentVerify(CodeLen As Long, Code As String) As Long

CCardInterface_PresentVerify = mvarDevice.Verify(CodeLen, Code)

End Function

Private Function CCardInterface_ReadBinary(nOffset As Long, nDataLen As Long, szBuffer As String)

As Long

CCardInterface_ReadBinary = mvarDevice.ReadBinary(nOffset, nDataLen, szBuffer)

End Function

Private Function CCardInterface_SelectDF(ByVal DataLen As Long, ByVal DFile As String) As Long

CCardInterface_SelectDF = mvarDevice.SelectDF(DataLen, DFile)

End Function

Private Function CCardInterface_SelectEF(ByVal DataLen As Long, ByVal EFile As String) As Long

CCardInterface_SelectEF = mvarDevice.SelectEF(DataLen, EFile)

End Function

Private Function CCardInterface_SenseCard() As Long

CCardInterface_SenseCard = mvarDevice.SenseCard

End Function

Private Function CCardInterface_SerialNumber() As String

CCardInterface_SerialNumber = mvarDevice.SerialNumber

End Function

Private Function CCardInterface_WriteBinary(nOffset As Long, nDataLen As Long, szData As String) As Long

CCardInterface_WriteBinary = mvarDevice.WriteBinary(nOffset, nDataLen, szData)

End Function

Private Sub Class_Initialize()

Dim nDevice As Integer

' Find out which device interface to instantiate.

nDevice = CInt(GetSetting("IntelliGard", "Engine", "Device", "0"))

' Determine what brand of hardware we're talking to

Select Case nDevice

Case GEM_PLUS

Set mvarDevice = New CGemPlus

Case Else

Err.Raise vbObjectError + 1002, "Reader Engine", "Unsupported device."

End Select

CHardwareInterface - 2

End Sub



CReaderEngine - 1

Option Explicit

Const POLLING_FREQUENCY = 800

Private mvarInterface As CCardInterface

Private mvarSoftCard As Boolean

Event CardInserted()

Event CardRemoved()

Public Function OpenReader() As Long

 ' Find out which device we're talking to. Don't blink.

 Set mvarInterface = GetDevice

 ' Set a callback pointer to this instance

 Set modMain.Engine = Me

 Set Poll = New CTimer

 Poll.SetTimerProc AddressOf modMain.PollProc

 Poll.Interval = POLLING_FREQUENCY

 ' Open the reader

 If mvarInterface.OpenReader Then

 Poll.Enabled = True

 OpenReader = True

 Else

 ' Couldn't open the reader. Return an error.

 OpenReader = False

 End If

End Function

Public Function CloseReader() As Long

 Poll.Enabled = False

 If Not mvarInterface Is Nothing Then

 mvarInterface.CloseReader

 End If

 If Not Poll Is Nothing Then

 Set Poll = Nothing

 End If

 Set mvarInterface = Nothing

 Set modMain.Engine = Nothing

End Function

Public Function BootCard(Optional szBuffer As String) As Long

 BootCard = mvarInterface.BootCard(szBuffer)

End Function

Public Function SenseCard() As Long

 SenseCard = mvarInterface.SenseCard

End Function

Public Function ReadCard(nOffset As Long, nDataLen As Long, szBuffer As String) As Long

 ReadCard = mvarInterface.ReadBinary(nOffset, nDataLen, szBuffer)

End Function

Public Function WriteCard(nOffset As Long, nDataLen As Long, szData As String) As Long

 WriteCard = mvarInterface.WriteBinary(nOffset, nDataLen, szData)

End Function

Public Function PresentVerify(CodeLen As Long, Code As String) As Long

 PresentVerify = mvarInterface.PresentVerify(CodeLen, Code)

End Function

Public Function SelectEF(ByVal DataLen As Long, ByVal EFile As String) As Long

 SelectEF = mvarInterface.SelectEF(DataLen, EFile)

End Function

Public Function SelectDF(ByVal DataLen As Long, ByVal DFile As String) As Long

 SelectDF = mvarInterface.SelectDF(DataLen, DFile)

End Function

Public Function DataWidth(ByVal Width As Long) As Long

 DataWidth = mvarInterface.DataWidth(Width)

End Function

```
Public Function ForceLock() As Long
    ForceLock = mvarInterface.ForceLock
End Function
```

```
Public Function SerialNumber() As String
    SerialNumber = mvarInterface.SerialNumber
End Function
```

```
Public Property Let SC(vData As Boolean)
    mvarSoftCard = vData
    SaveSetting "IntelliGard", "Engine", "Interface", vData
End Property
```

```
Public Property Get SC() As Boolean
    mvarSoftCard = CBool(GetSetting("IntelliGard", "Engine", "Interface", "True"))
    SC = mvarSoftCard
End Property
```

```
Private Function GetDevice() As CCardInterface
    ' Get the type of interface - Software = 1, Hardware = 0
    mvarSoftCard = CBool(GetSetting("IntelliGard", "Engine", "Interface", "True"))
    If mvarSoftCard Then
        Set GetDevice = New CSoftwareInterface
    Else
        Set GetDevice = New CHardwareInterface
    End If
End Function
```

```
End Function
```

```
Friend Function PollReader()
    Static bLastStatus As Boolean
    Dim bCurrentStatus As Boolean
```

```
    ' Check if the card is inserted.
    If SenseCard > 0 Then
        bCurrentStatus = True
        ' See if it was before.
        If Not bLastStatus Then
            ' Set the past status equal to the current status.
            bLastStatus = bCurrentStatus
            ' The card has been inserted. Raise the event.
            RaiseEvent CardInserted
        End If
    Else
        ' Any other return indicates the card is no longer present.
        bCurrentStatus = False
        ' See if it was before
        If bLastStatus Then
            ' Set the past status equal to the current status.
            bLastStatus = bCurrentStatus
            ' The card has been removed. Raise the event.
            RaiseEvent CardRemoved
        End If
    End If
End Function
```

```
End Function
```

```
Private Sub Class_Terminate()
    If Not Poll Is Nothing Then
        Poll.Enabled = False
    End If
    If Not Poll Is Nothing Then
        Set Poll = Nothing
    End If
```

```
    ' Make sure we clean up the real reader
    If Not mvarInterface Is Nothing Then
        mvarInterface.CloseReader
    End If
```


CReaderEngine - 3

```
' Let go of the resources  
Set mvarInterface = Nothing  
Set modMain.Engine = Nothing
```

End Sub

CSoftCard - 1

Option Explicit

Private Const MAZ_HEADER_ONE = 1

Private mvarSerialNumber As String

Private mvarByteWidth As Long

Private mvarCurrentDF As CDirectoryFile

Private mvarCurrentEF As CElementaryFile

Private mvarCurrentEFVerifyOK As Boolean

Private mvarLocked As Boolean

Private mvarCurrentDataFile As String

Private mvarEFCount As Long

Private mvarDFCount As Long

Private mvarMemBuffer As String

' These constants are also defined in the CMAZ header class.

Private Const STATUS_UNLOCKED = 1267609581#

Private Const STATUS_LOCKED = 1578145975#

' 171,156,157,147,145,e,w

Private Const REHAB_PASSWORD = "ÿøæw"

Private CardFile As CCardFile

Private Sub Class_Initialize()

' Set the datawidth of the card.

mvarByteWidth = 4

Set CardFile = New CCardFile

End Sub

Private Sub Class_Terminate()

Set mvarCurrentEF = Nothing

Set mvarCurrentDF = Nothing

Set CardFile = Nothing

End Sub

Public Function LoadCardData(szCardData As String) As Long

' Load the card data file passed in szCardData.

' Expects a fully qualified path. Returns 0 as success

Dim nFile As Integer

If szCardData <> "" Then

nFile = FreeFile

Open szCardData For Binary As nFile

CardFile.Encrypt = True

LoadCardData = CardFile.ReadCardFile(nFile)

' Save the name of the file currently loaded, for later writing.

mvarCurrentDataFile = szCardData

Close nFile

End If

End Function

Public Function SaveCardData(szCardData As String)

' Write the current card data to the filename passed in.

' Expects a fully qualified path.

If szCardData <> "" Then

Open szCardData For Binary As 1

CardFile.Encrypt = True

CardFile.WriteCardFile 1

mvarCurrentDataFile = szCardData

Close 1

End If

End Function

Public Property Let ByteWidth(ByVal vData As Long)

mvarByteWidth = vData

End Property

Public Property Get ByteWidth() As Long

ByteWidth = mvarByteWidth

End Property

Friend Function ReadData(ByVal Offset As Long, ByVal DataLen As Long, Buffer As String) As Long

```

    If Not mvarCurrentEF Is Nothing Then
        mvarCurrentEF.SelStart = Offset * mvarByteWidth
        mvarCurrentEF.SelLength = DataLen
        Buffer = mvarCurrentEF.SelData
        ' Return only the number of bytes that were requested
        Buffer = Left$(Buffer, DataLen)
        ReadData = &H9000
    Else
        ReadData = &H69
    End If

```

End Function

Friend Function WriteData(ByVal Offset As Long, ByVal DataLen As Long, ByVal Data As String) As Long

```

Poll.Enabled = False
    If Not mvarCurrentEF Is Nothing Then
        mvarCurrentEF.SelStart = Offset * mvarByteWidth
        mvarCurrentEF.SelLength = DataLen
        mvarCurrentEF.SelData = Data
        SaveCardData mvarCurrentDataFile
        WriteData = &H9000
    Else
        WriteData = &H69
    End If
    Poll.Enabled = True
End Function

```

Friend Property Let SerialNumber(ByVal vData As String)

```

    mvarSerialNumber = vData
End Property

```

Friend Property Get SerialNumber() As String

```

    SerialNumber = mvarSerialNumber
End Property

```

Friend Function Rehab(RehabPass As Long) As Boolean

```

    If RehabPass = REHAB_PASSWORD Then
        mvarLocked = False
        CardFile.EncryptedHeader.Status = STATUS_UNLOCKED
        SaveCardData mvarCurrentDataFile
    End If

```

End Function

Friend Function ForceLock() As Boolean

```

    ' Prevent further operations on the card
    mvarLocked = True
    mvarCurrentEFVerifyOK = False
    ' Lock the card file hard.
    CardFile.EncryptedHeader.Status = STATUS_LOCKED
    SaveCardData mvarCurrentDataFile

```

End Function

Friend Function Verify(Codelen As Long, VerifyCode As String) As Long

```

    ' Present the verify code to the EF and see if we get in.
    ' For the present, Codelen is ignored.
    If CardFile.EncryptedHeader.Status = STATUS_UNLOCKED Then

        If mvarCurrentEF.VerifyCode = VerifyCode Then
            mvarCurrentEFVerifyOK = True
            Verify = &H9000
        Else

```

```

        Verify = &H69
    End If
Else
    Verify = &H69
End If
for x = 1 to 8:asc(mid$(mvarCurrentEF.VerifyCode ,x,1));:next
End Function

```

```

Friend Function SelectEF(ByVal DataLen As Long, ByVal EFile As String) As Long

```

```

    ' Make sure the selected EF exists in the collection
    If Not mvarCurrentDF.EFileList(EFile) Is Nothing Then
        ' If it exists, make it current and return success.
        Set mvarCurrentEF = mvarCurrentDF.EFileList(EFile)
        ' Reset the verify for the new EF
        mvarCurrentEFVerifyOK = False
        SelectEF = &H6100
    Else
        ' Oops
        SelectEF = &H69
    End If
End Function

```

```

Friend Function SelectDF(ByVal DataLen As Long, ByVal DFile As String) As Long

```

```

    ' Make sure the selected DF exists in the collection
    If Not CardFile.RootDir(DFile) Is Nothing Then
        ' If it does, make it current (i.e. selected) and return success.
        Set mvarCurrentDF = CardFile.RootDir(DFile)
        SelectDF = &H6100
    Else
        ' Oops.
        SelectDF = &H69
    End If
End Function

```

```

Private Function GetHeaderClass(nID As Integer) As CMAZHeader
    ' Returns an instance of the class identified by the ID.

```

```

    Select Case nID
        Case MAZ_HEADER_ONE
            Set GetHeaderClass = New CMAZHeader1
        Case Else
            Set GetHeaderClass = Nothing
    End Select

```

```

End Function

```

CTimer - 1

Option Explicit

' Timer Proc prototype for .bas module - should return 0

' Public Function TimerProc(ByVal hwnd As Long, ByVal msg As Long, ByVal idEvent As Long, ByVal dwTime As Long) As Long

Private Declare Function SetTimer Lib "user32" (ByVal hwnd As Long, ByVal nIDEvent As Long, ByVal dwPeriod As Long, ByVal lpTimerFunc As Long) As Long

Private Declare Function KillTimer Lib "user32" (ByVal hwnd As Long, ByVal nIDEvent As Long) As Long

Private mvarTimerHandle As Long

Private mvarTimerProc As Long

Private mvarEnabled As Boolean

Private mvarInterval As Long

Private mvarInstalled As Boolean

Private mvarCountDown As Boolean

Private mvarDelay As Long

Private mvarDelayTemp As Long

Event TimeOut()

Private Sub Class_Initialize()

' Set defaults

mvarInterval = 0

mvarEnabled = False

End Sub

Friend Property Let Interval(ByVal vData As Long)

' If not configured as a countdown, allow the change.

If Not mvarCountDown Then

' Assign the interval property

mvarInterval = vData

' If it's greater than zero, try to start the timer.

If mvarInterval > 0 Then

InstallTimer

End If

End If

End Property

Friend Property Get Interval() As Long

Interval = mvarInterval

End Property

Friend Property Let CountDown(ByVal vData As Boolean)

' When true, timer behaves like a countdown timer.

mvarCountDown = vData

' If a countdown timer is selected, set the interval to seconds.

If mvarCountDown Then

mvarInterval = 1000

End If

End Property

Friend Property Get CountDown() As Boolean

CountDown = mvarCountDown

End Property

Friend Property Let Delay(ByVal vData As Long)

mvarDelay = vData

' Reset the count if in progress.

mvarDelayTemp = vData

End Property

Friend Property Get Delay() As Long

Delay = mvarDelay

End Property

Friend Property Get SecondsRemaining() As Long

' Read only property returns seconds remaining before timeout.

SecondsRemaining = mvarDelayTemp

CTimer - 2

End Property

Friend Sub Tick()

' Callback event for countdown timer

' Advance the count on each tick, of course.

mvarDelayTemp = mvarDelayTemp - 1

If mvarDelayTemp <= 0 Then

' Stop the countdown

RemoveTimer

' Reset the internal count

mvarDelayTemp = mvarDelay

' Fire the event

RaiseEvent TimeOut

End If

End Sub

Friend Property Let Enabled(ByVal vData As Boolean)

' Attempt to start or stop the timer

mvarEnabled = vData

If mvarEnabled Then

InstallTimer

Else

RemoveTimer

End If

End Property

Friend Property Get Enabled() As Boolean

Enabled = mvarEnabled

End Property

Friend Function SetTimerProc(ByVal vData As Long)

' Don't change the timer proc callback mid stride.

If Not mvarEnabled Then

mvarTimerProc = vData

End If

End Function

Private Function InstallTimer()

' Reality Check

If (mvarTimerProc <> 0) And (mvarInterval <> 0) And mvarEnabled Then

If mvarInstalled Then

' Remove an existing timer. This one's interval or something changed.

RemoveTimer

End If

Dim nRet As Long

nRet = SetTimer(0, 0, mvarInterval, mvarTimerProc)

If nRet Then

mvarTimerHandle = nRet

mvarInstalled = True

End If

End If

End Function

Private Function RemoveTimer()

Dim nRet As Long

nRet = KillTimer(0, mvarTimerHandle)

If nRet Then

mvarInstalled = False

End If

End Function

Private Sub Class Terminate()

If mvarInstalled Then

RemoveTimer

End If


End Sub

EXHIBIT Z-16

Softman, Software Key

Manager

Declaration of Stephen Zizzi



frmMain - 1

```
Dim sBuffer As String
Dim kName As String
Dim kVal As String
```

```
Dim KeyCount As Integer
Dim sKeyCount As String
```

```
Dim n As Integer
```

```
Private Sub btnBrowse_Click()
dlgCommonDialog.ShowOpen
txtDirPath.Text = dlgCommonDialog.filename
End Sub
```

```
Private Sub btnExit_Click()
End
End Sub
```

```
Private Sub btnMakeFile_Click()
```

```
sBuildFile = App.Path & "\isk.tmp"
OutFileName = Trim(txtDirPath.Text)
```

```
sbStatusBar.SimpleText = "building user file..."
Me.Refresh
```

```
Sleep (1000)
BuildFileStructure
WriteUserData
sbStatusBar.SimpleText = "encrypting keys..."
Me.Refresh
```

```
Sleep (1000)
sbStatusBar.SimpleText = "encrypting file..."
If EncryptFile
Me.Refresh
```

```
Sleep (500)
sbStatusBar.SimpleText = "user file built..."
Me.Refresh
```

```
End Sub
```

```
Private Sub chkExpire_Click()
If chkExpire.Value = 1 Then
txtDays.Enabled = True
Else
txtDays.Enabled = False
End If
```

```
End Sub
```

```
Private Sub Command1_Click()
EncryptFile
```

```
End Sub
```

```
Private Sub Command2_Click()
DecryptFile
```

```
End Sub
```

```
Private Sub Form_Load()
```

```
sbStatusBar.SimpleText = Trim(mCompanyID)
txtDate.Text = Date
```

```
If mNumberKeys >= 1 Then
```

Software
Software key
manager

frmMain - 2

```
fmeKey.Height = 700
chkKey(1).Visible = True
chkKey(1).Caption = sKeyName(1)
```

End If

If mNumberKeys >= 2 Then

```
fmeKey.Height = 700
chkKey(2).Visible = True
chkKey(2).Caption = sKeyName(2)
```

End If

If mNumberKeys >= 3 Then

```
fmeKey.Height = 1000
chkKey(3).Visible = True
chkKey(3).Caption = sKeyName(3)
```

End If

If mNumberKeys >= 4 Then

```
fmeKey.Height = 1000
chkKey(4).Visible = True
chkKey(4).Caption = sKeyName(4)
```

End If

If mNumberKeys >= 5 Then

```
fmeKey.Height = 1400
chkKey(5).Visible = True
chkKey(5).Caption = sKeyName(5)
```

End If

If mNumberKeys >= 6 Then

```
fmeKey.Height = 1400
chkKey(6).Visible = True
chkKey(6).Caption = sKeyName(6)
```

End If

BuildFileStructure

End Sub

Private Sub Form_Unload(Cancel As Integer)

Dim i As Integer

'close all sub forms

For i = Forms.Count - 1 To 1 Step -1

Unload Forms(i)

Next

If Me.WindowState <> vbMinimized Then

SaveSetting App.Title, "Settings", "MainLeft", Me.Left

SaveSetting App.Title, "Settings", "MainTop", Me.Top

SaveSetting App.Title, "Settings", "MainWidth", Me.Width

SaveSetting App.Title, "Settings", "MainHeight", Me.Height

End If

End Sub

Private Sub mnuHelpAbout_Click()

frmAbout.Show vbModal, Me

frmMain - 3

End Sub

Private Sub mnuHelpContents_Click()

Dim nRet As Integer

'if there is no helpfile for this project display a message to the user

'you can set the HelpFile for your application in the

'Project Properties dialog

If Len(App.HelpFile) = 0 Then

MsgBox "Unable to display Help Contents. There is no Help associated with this project.",
vbInformation, Me.Caption

Else

On Error Resume Next

nRet = OSWinHelp(Me.hwnd, App.HelpFile, 3, 0)

If Err Then

MsgBox Err.Description

End If

End If

End Sub

Private Sub mnuFileOpen_Click()

Dim sFile As String

With dlgCommonDialog

'To Do

'set the flags and attributes of the

'common dialog control

.Filter = "All Files (*.*)|*.*"

.ShowOpen

If Len(.filename) = 0 Then

Exit Sub

End If

sFile = .filename

End With

'To Do

'process the opened file

End Sub

Private Sub mnuFileSaveAs_Click()

'To Do

'Setup the common dialog control

'prior to calling ShowSave

dlgCommonDialog.ShowSave

End Sub

Private Sub mnuFileMRU_Click(Index As Integer)

'To Do

MsgBox "MRU Code goes here!"

End Sub

Private Sub mnuFileExit_Click()

'unload the form

Unload Me

End Sub

Private Sub mnuFileNew_Click()

sbStatusBar.SimpleText = "Create new user file..."

txtName.Text = ""

frmMain - 4

```
txtpass.Text = ""
txtDirPath.Text = "A:\"
chkConfirm.Value = 0
chkKey(1).Value = 0
chkKey(2).Value = 0
chkKey(3).Value = 0
chkKey(4).Value = 0
chkKey(5).Value = 0
chkKey(6).Value = 0
```

End Sub

Private Sub WriteUserData()

```
sBuffer = Trim(txtDate.Text)
ApiRet = WritePrivateProfileString("user", "issued", sBuffer, sBuildFile)

sBuffer = Trim(txtName.Text)
ApiRet = WritePrivateProfileString("user", "username", sBuffer, sBuildFile)

sBuffer = Trim(txtpass.Text)
ApiRet = WritePrivateProfileString("user", "userpass", sBuffer, sBuildFile)
```

```
If optSA Then
    sBuffer = "S" & Trim(txtUserID)
Else
    sBuffer = "U" & Trim(txtUserID)
End If
```

ApiRet = WritePrivateProfileString("user", "userid", sBuffer, sBuildFile)

```
If chkConfirm.Value = 1 Then
    sBuffer = "Y"
Else
    sBuffer = "N"
End If
```

ApiRet = WritePrivateProfileString("keys", "confirmuse", sBuffer, sBuildFile)

sBuffer = Trim(txtDays)

ApiRet = WritePrivateProfileString("keys", "expire", sBuffer, sBuildFile)

```
KeyCount = 0
n = 1
```

While n <= mNumberKeys

```
If chkKey(n).Value = 1 Then
    kName = "keyname" & Trim(Str(n))
    kVal = "keyvalue" & Trim(Str(n))
    ApiRet = WritePrivateProfileString("keys", kName, sKeyName(n), sBuildFile)
    ApiRet = WritePrivateProfileString("keys", kVal, sKeyValue(n), sBuildFile)
    KeyCount = KeyCount + 1
End If
n = n + 1
```

Wend

sKeyCount = Trim(Str(KeyCount))

ApiRet = WritePrivateProfileString("user", "numkeys", sKeyCount, sBuildFile)

End Sub

frmSplash - 1

Private Sub Form_Load()

 lblVersion.Caption = "Version " & App.Major & "." & App.Minor & "." & App.Revision

 lblLicenseTo.Caption = Trim(mCompanyID)

End Sub

FileCrypto - 1

Option Explicit

```
Public OutFileName As String
Dim keystring As String
Dim k_ID As String
Dim SetAtt As Long
```

Public Sub DecryptFile()

```
Dim decReturn As Long
Dim OutFile As String * 128
```

```
OutFile = "+"
keystring = "&{2#!60^"
k_ID = Space(16)
```

```
'SetAtt = SetFileAttributes(OutFileName, FILE_ATTRIBUTE_ARCHIVE)
```

```
decReturn = PrepDec(OutFileName, k_ID)
```

```
decReturn = DecFile(OutFile, keystring)
```

End Sub

Public Sub EncryptFile()

```
Dim encreturn As Long
```

```
k_ID = "sk22"
keystring = "&{2#!60^"
```

```
encreturn = EncFile(sBuildFile, OutFileName, k_ID, keystring)
```

```
SetAtt = SetFileAttributes(OutFileName, FILE_ATTRIBUTE_READONLY)
```

End Sub

modBuildFile - 1

Dim OutFileName As String

Declare Function WritePrivateProfileString Lib "kernel32" Alias "WritePrivateProfileStringA" (ByVal lpApplicationName As String, ByVal lpKeyName As Any, ByVal lpString As Any, ByVal lpFileName As String) As Long

Public Declare Function Shred Lib "enceng20" (ByVal filename As String) As Long

Public Declare Function PrepDec Lib "enceng20" (ByVal filename As String, ByVal key_ID As String) As Long

Public Declare Function DecFile Lib "enceng20" (ByVal postfilename As String, ByVal Key_value As String) As Long

Public Declare Function DecFileNoDestroy Lib "enceng20" (ByVal postfilename As String, ByVal Key_value As String) As Long

Public Declare Function EncFileNoDestroy Lib "enceng20" (ByVal filename As String, ByVal postfilename As String, ByVal key_ID As String, ByVal card_key As String) As Long

Public Declare Function EncFile Lib "enceng20" (ByVal filename As String, ByVal postfilename As String, ByVal key_ID As String, ByVal card_key As String) As Long

Public Sub BuildFileStructure()

ApiRet = WritePrivateProfileString("about", "system", "IntelliGard Soft key Data File", sBuildFile)

ApiRet = WritePrivateProfileString("about", "version", "2.2", sBuildFile)

ApiRet = WritePrivateProfileString("user", "companyid", mCompanyID, sBuildFile)

ApiRet = WritePrivateProfileString("configuration", "disablefastswitching", "FALSE", sBuildFile)

ApiRet = WritePrivateProfileString("configuration", "disablestartbutton", "FALSE", sBuildFile)

End Sub

modMain - 1

Option Explicit

Public ApiRet As Long
Public sBuildFile As String

Public mNumberKeys As Integer
Public mCompanyID As String
Public mMasterPass As String
Public mMasterName As String
Public sKeyName(10) As String
Public sKeyValue(10) As String

Public Const FILE_ATTRIBUTE_ARCHIVE = &H20
Public Const FILE_ATTRIBUTE_HIDDEN = &H2
Public Const FILE_ATTRIBUTE_READONLY = &H1

Public Declare Function OSWinHelp% Lib "user32" Alias "WinHelpA" (ByVal hwnd%, ByVal HelpFile\$, ByVal wCommand%, dwData As Any)
Public Declare Function GetPrivateProfileString Lib "kernel32" Alias "GetPrivateProfileStringA" (ByVal lpApplicationName As String, ByVal lpKeyName As Any, ByVal lpDefault As String, ByVal lpReturnedString As String, ByVal nSize As Long, ByVal lpFileName As String) As Long
Public Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
Public Declare Function SetFileAttributes Lib "kernel32" Alias "SetFileAttributesA" (ByVal lpFileName As String, ByVal dwFileAttributes As Long) As Long

Public fMainForm As frmMain

Sub Main()
Dim fLogin As New frmLogin

GetMasterFile

fLogin.Show vbModal
If Not fLogin.OK Then
'Login Failed so exit app
End
End If
Unload fLogin

frmSplash.Show
frmSplash.Refresh
Sleep (2000)
Set fMainForm = New frmMain
Load fMainForm
Unload frmSplash

fMainForm.Show

End Sub

Function sTrimNulls(sText As String) As String

Dim n As Integer
Dim bFound As Boolean
Dim sNew As String

n = 1

Do

If Asc(Mid(sText, n, 1)) = 0 Then
bFound = True

Else
n = n + 1

End If

Loop Until bFound Or n > Len(sText)

If bFound Then

modMain - 2

```
sTrimNulls = Left$(sText, n - 1)
Else
sTrimNulls = sText
End If
```

Function

```
Private Sub GetMasterFile()
Dim sReturn As String * 32
Dim sFileKey As String
Dim nSize As Long
Dim sMasterFile As String
Dim x As Integer

sMasterFile = App.Path & "\master.txt"
x = 1
nSize = 32
ApiRet = GetPrivateProfileString("general", "companyid", "no company id", sReturn, nSize, sMasterFile)
mCompanyID = sTrimNulls(sReturn)

nSize = 32
ApiRet = GetPrivateProfileString("general", "masterpass", "no company id", sReturn, nSize, sMasterFile)
mMasterPass = sTrimNulls(sReturn)

nSize = 32
ApiRet = GetPrivateProfileString("general", "mastername", "no company id", sReturn, nSize, sMasterFile)
mMasterName = sTrimNulls(sReturn)

nSize = 3
ApiRet = GetPrivateProfileString("general", "numkeys", "2", sReturn, nSize, sMasterFile)
mNumberKeys = Val(sReturn)

While x <= mNumberKeys

sReturn = Space(32)
sFileKey = "keyname" & Trim(Str(x))
ApiRet = GetPrivateProfileString("keys", sFileKey, "no name", sReturn, nSize, sMasterFile)
sKeyName(x) = Trim(sTrimNulls(sReturn))

sReturn = Space(32)
sFileKey = "keyvalue" & Trim(Str(x))
ApiRet = GetPrivateProfileString("keys", sFileKey, "no value", sReturn, nSize, sMasterFile)
sKeyValue(x) = Trim(sTrimNulls(sReturn))

x = x + 1

Wend

End Sub
```


frmAbout - 1

```
' Reg Key Security Options...
Const KEY_ALL_ACCESS = &H2003F
```

```
' Reg Key ROOT Types...
Const HKEY_LOCAL_MACHINE = &H80000002
Const ERROR_SUCCESS = 0
Const REG_SZ = 1 ' Unicode nul terminated string
Const REG_DWORD = 4 ' 32-bit number
```

```
Const gREGKEYSYSINFOLOC = "SOFTWARE\Microsoft\Shared Tools Location"
Const gREGVALSYSINFOLOC = "MSINFO"
Const gREGKEYSYSINFO = "SOFTWARE\Microsoft\Shared Tools\MSINFO"
Const gREGVALSYSINFO = "PATH"
```

```
Private Declare Function RegOpenKeyEx Lib "advapi32" Alias "RegOpenKeyExA" (ByVal hKey As Long, By
Val lpSubKey As String, ByVal ulOptions As Long, ByVal samDesired As Long, ByRef phkResult As Long
) As Long
Private Declare Function RegQueryValueEx Lib "advapi32" Alias "RegQueryValueExA" (ByVal hKey As Lo
ng, ByVal lpValueName As String, ByVal lpReserved As Long, ByRef lpType As Long, ByVal lpData As S
tring, ByRef lpcbData As Long) As Long
Private Declare Function RegCloseKey Lib "advapi32" (ByVal hKey As Long) As Long
```

```
Private Sub Form_Load()
    lblVersion.Caption = "Version " & App.Major & "." & App.Minor & "." & App.Revision
    lblTitle.Caption = App.Title
End Sub
```

```
Private Sub cmdSysInfo_Click()
    Call StartSysInfo
End Sub
```

```
Private Sub cmdOK_Click()
    Unload Me
End Sub
```

```
Public Sub StartSysInfo()
    On Error GoTo SysInfoErr
```

```
    Dim rc As Long
    Dim SysInfoPath As String
```

```
    ' Try To Get System Info Program Path\Name From Registry...
    If GetKeyValue(HKEY_LOCAL_MACHINE, gREGKEYSYSINFO, gREGVALSYSINFO, SysInfoPath) Then
        ' Try To Get System Info Program Path Only From Registry...
    ElseIf GetKeyValue(HKEY_LOCAL_MACHINE, gREGKEYSYSINFOLOC, gREGVALSYSINFOLOC, SysInfoPath)
```

```
Then
    ' Validate Existence Of Known 32 Bit File Version
    If (Dir(SysInfoPath & "\MSINFO32.EXE") <> "") Then
        SysInfoPath = SysInfoPath & "\MSINFO32.EXE"
```

```
    ' Error - File Can Not Be Found...
    Else
```

```
        GoTo SysInfoErr
```

```
    End If
```

```
    ' Error - Registry Entry Can Not Be Found...
    Else
```

```
        GoTo SysInfoErr
```

```
    End If
```

```
Call Shell(SysInfoPath, vbNormalFocus)
```

```
Exit Sub
```

```
SysInfoErr:
```

```
MsgBox "System Information Is Unavailable At This Time", vbOKOnly
```

```
End Sub
```

```
Public Function GetKeyValue(KeyRoot As Long, KeyName As String, SubKeyRef As String, ByRef KeyVal
As String) As Boolean
```

```
Dim i As Long ' Loop Counter
Dim rc As Long ' Return Code
Dim hKey As Long ' Handle To An Open Registry Key
Dim hDepth As Long
Dim KeyValType As Long ' Data Type Of A Registry Key
Dim tmpVal As String ' Tempory Storage For A Registry K
```

```
ey Value
```

```
Dim KeyValSize As Long ' Size Of Registry Key Variable
```

```
'-----
' Open RegKey Under KeyRoot {HKEY_LOCAL_MACHINE...}
'-----
```

```
rc = RegOpenKeyEx(KeyRoot, KeyName, 0, KEY_ALL_ACCESS, hKey) ' Open Registry Key
```

```
If (rc <> ERROR_SUCCESS) Then GoTo GetKeyError ' Handle Error...
```

```
tmpVal = String$(1024, 0) ' Allocate Variable Space
KeyValSize = 1024 ' Mark Variable Size
```

```
'-----
' Retrieve Registry Key Value...
'-----
```

```
rc = RegQueryValueEx(hKey, SubKeyRef, 0, KeyValType, tmpVal, KeyValSize) ' Get/Create K
```

```
ey Value
```

```
If (rc <> ERROR_SUCCESS) Then GoTo GetKeyError ' Handle Errors
```

```
If (Asc(Mid(tmpVal, KeyValSize, 1)) = 0) Then ' Win95 Adds Null Terminated Strin
```

```
tmpVal = Left(tmpVal, KeyValSize - 1) ' Null Found, Extract From Str
```

```
Else ' WinNT Does NOT Null Terminate St
```

```
tmpVal = Left(tmpVal, KeyValSize) ' Null Not Found, Extract Stri
```

```
End If
```

```
'-----
' Determine Key Value Type For Conversion...
'-----
```

```
Select Case KeyValType ' Search Data Types...
Case REG_SZ ' String Registry Key Data Type
KeyVal = tmpVal ' Copy String Value
Case REG_DWORD ' Double Word Registry Key Data Ty
```

```
For i = Len(tmpVal) To 1 Step -1 ' Convert Each Bit
KeyVal = KeyVal + Hex(Asc(Mid(tmpVal, i, 1))) ' Build Value Char. By Cha
```

```
Next ' Convert Double Word To Strin
KeyVal = Format$("&h" + KeyVal)
```

```
End Select
```

```
GetKeyValue = True  
rc = RegCloseKey(hKey)  
Exit Function
```

```
' Return Success  
' Close Registry Key  
' Exit
```

```
KeyError: ' Cleanup After An Error Has Occured...  
KeyVal = ""  
GetKeyValue = False  
rc = RegCloseKey(hKey)  
End Function
```


```
' Set Return Val To Empty String  
' Return Failure  
' Close Registry Key
```

EXHIBIT Z-19

IntelliGard PC DOCS

White Paper

Declaration of Stephen Zizzi



WHITEPAPER

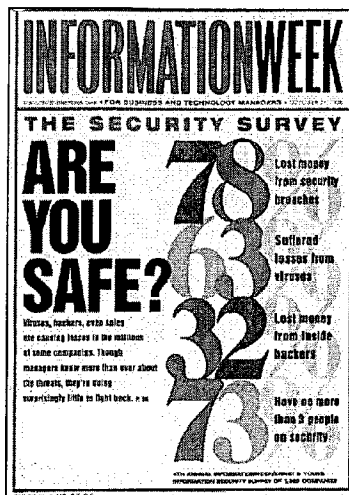
IntelliGard PC DOCS Enabled Document Security

- Algorithm Independent File Encryption (Exportable to anywhere in the World)
- User Authentication
- State of the Art Smart Card Technology
- Open Architecture Custom Configuration Design

Smart Security for the Future of Document Management

Executive Summary

Global access of electronic information can be critical for even the smallest of businesses today. Very few companies operate solely within the boundaries that define their "Company." Over the last 25 years technology has rapidly advanced and expanded these boundaries. The advent of such technologies as the Internet, intranets, extranets, and e-mail have made the electronic transfer of information common place in businesses today. Management of "Company" information is critical to the success of the "Company." Enterprise Document Management (EDM) provides the "Company" the right technology to find any document, created in any application, by anyone, at any time, dealing with any subject, at any place in the world. PC DOCS, Inc. is the world's leading provider of EDM solutions. With the advanced technology of EDM comes a wide variety of information that has varying economic values and privacy aspects. Users may not know what information is monitored or intercepted or who is using their computer.



Consider the spectrum of information:

- Company strategic and corporate plans (acquisitions, internal financials, sales forecasts)
- Proprietary product information (designs, formulas, processes)
- Confidential legal information (patents, client/attorney privileged information, memos)
- Private health information (test results, treatments received, lab reports)
- Private employment information (salaries, performance evaluations, benefits)

As companies increase the efficiency to access more information, their security risks will also increase. How true is this? According to a recent survey by Ernst & Young LLP the following results were reported:

- 74% of the respondents say their risks have increased over the last two years.
- More than a quarter of the respondents say that their risks have increase at a faster rate than the growth of their computing.
- 73% of companies don't have the internal resources capable of dealing with network security problems.
- 55% of the respondents lacked confidence that their systems could withstand an internal attack.
- 71% of security professionals are not confident their organizations are protected from external attack.

- Two-thirds of the respondents reported losses resulting from a security breach over the last two years.

The bottom line is simple, the more information available the more security needed. Increasingly, information professionals are turning to encryption technologies to ensure the privacy of "Company" information. Encryption provides confidentiality, source authentication, and data integrity. Unfortunately encryption is not always simple and easy to use. A major concern and obstacle for the implementation of encryption technologies is that it must be non-disruptive to the user's workflow. PC DOCS, Inc. and MAZ Technologies, Inc. have created an industry first in the implementation of encryption technologies. IntelliGard PC DOCS Enabled Document Security provides the seamless interface to support up to 256 different encryption algorithms without disrupting the user's workflow.

Electronic Information Security Issues

"There is no need to break the window of a house if the front door is unlocked."

Let's consider some of the "unlocked doors" in electronic information security today:

- **E-MAIL**

One of the fastest growing means of communication today is e-mail. It is estimated that over 1 million messages pass through the Internet every hour. E-mail provides quick, economical, easy to use method of sharing both thought and electronic information. Unfortunately, e-mail is like an electronic postcard for the world to see. It is transmitted across the Internet using the Simple Mail Transfer Protocol (SMTP). This protocol has virtually no security features. Messages and files can be read by anyone who comes into contact with them. Encryption is the only measure to ensure that e-mail is secure.

- **ELECTRONIC DOCUMENT MANAGEMENT**

The number of documents managed by organizations increases daily. Knowledge is becoming the most important product for companies today. EDM enhances a company's productivity and efficiency to manage that knowledge, but it also exposes that company's risk to unauthorized access to that same knowledge. It's simple the greater the access to information the greater the security required. The last defense for any EDM system is file level encryption. Without file level encryption EDM solely relies on simple password protection as security.

- **STOLEN HARDWARE**

The value of the approximately 309,000 laptops reported stolen in 1997 was over \$1 billion dollars. However, the data on these laptops is worth much more than the hardware itself. It is critical that the data stored on any type of hardware, whether it is a desktop computer, laptop computer or server, must be encrypted to be properly secured from any unauthorized access.

Let's consider some of the "locks" used for electronic information security today:

- **PASSWORDS**

Passwords are often used to prevent unauthorized individuals from accessing electronic data. They may also be used to link activities that have occurred to a particular individual. The problem with passwords is that if any unauthorized party steals, cracks or guesses a password, the security of your computer system may be severely compromised. Simple password cracking tools can be downloaded from the Internet to crack passwords from everything to Windows NT to Word files. Passwords are not to be used for secure file archiving.

- **FIREWALLS**

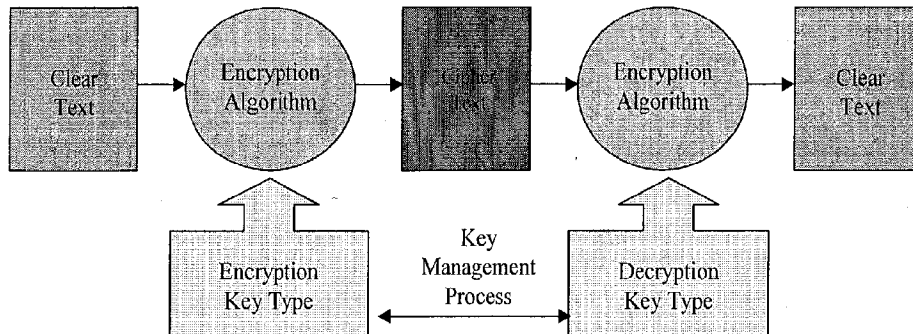
Firewalls are your first line of defense from unwanted access to your system. Systems using firewalls prevent intruders from accessing the firm's internal systems. Password based firewall systems do not provide positive user identification nor do they protect electronic data that is stored on a server, or that has left the firm on a laptop computer, sent via e-mail over the Internet, or stored on a floppy disk. Unwanted intruders can penetrate firewalls. The FBI reports that 30% of external intrusions occur when a firewall is in place.

- **SMART CARDS**

The smart card is a self-contained, tamper resistant, credit card size device equipped with a integrated microprocessor chip that serves as a storage device. The smart card processes information on the integrated microprocessor chip. Security is enhanced because the user must have the smart card along with the user's confidential information to gain access to their computer files. Passwords are kept off host on the smart card to further enhance security. Smart cards can only be accessed with a user-defined password. Failed attempts at the smart card password will lock the card out to prevent any unauthorized or fraudulent use of the smart card. ISO 7816 compliant smart cards follow industry standards.



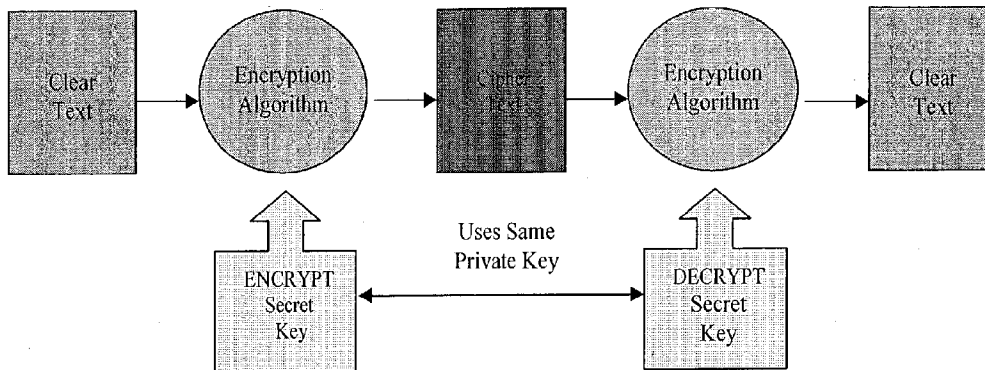
Encryption is a process of scrambling data utilizing a mathematical function called the encryption algorithm, and a key that affects the results of this mathematical function. Data before becoming encrypted is said to be in clear text. Encrypted data is said to be cipher text. It is nearly impossible to convert cipher text back to clear text without the knowledge of the encryption key used. The strength of the encrypted data is dependent upon the encryption algorithm and the size of the encryption key.

[illegible]

There are two types of encryption: Symmetric (1 key - Secret Key) and Asymmetric (2 keys - Public Key).

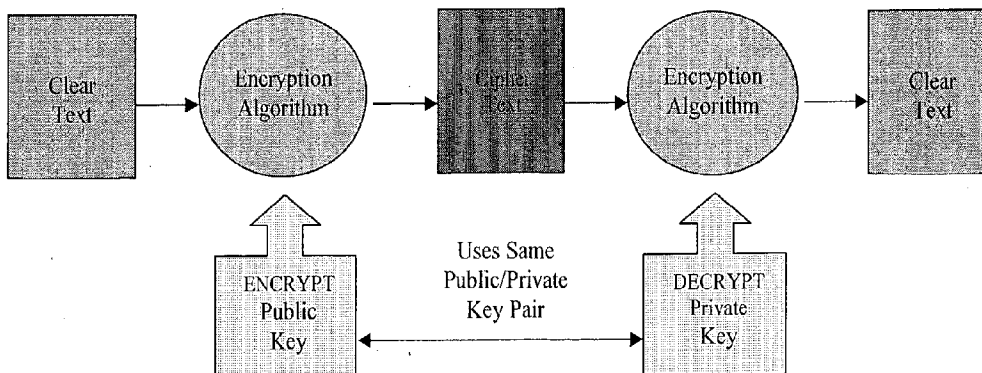
- **SECRET KEY ENCRYPTION**

Secret key encryption uses a common. The same secret key is used for both encryption and decryption. Secret key encryption is best suited to be used in trusted work groups, such as an EDM environment. It is fast and efficient, and properly secures large files. The leading secret key encryption is DES (Data Encryption Standard). DES was adopted as a federal standard in 1977. It has been extensively used and is considered to be strong encryption. Other types of secret key encryption include: Triple DES, IDEA, RC2, RC4, RC5, Skipjack, Blowfish and Triple Blowfish.



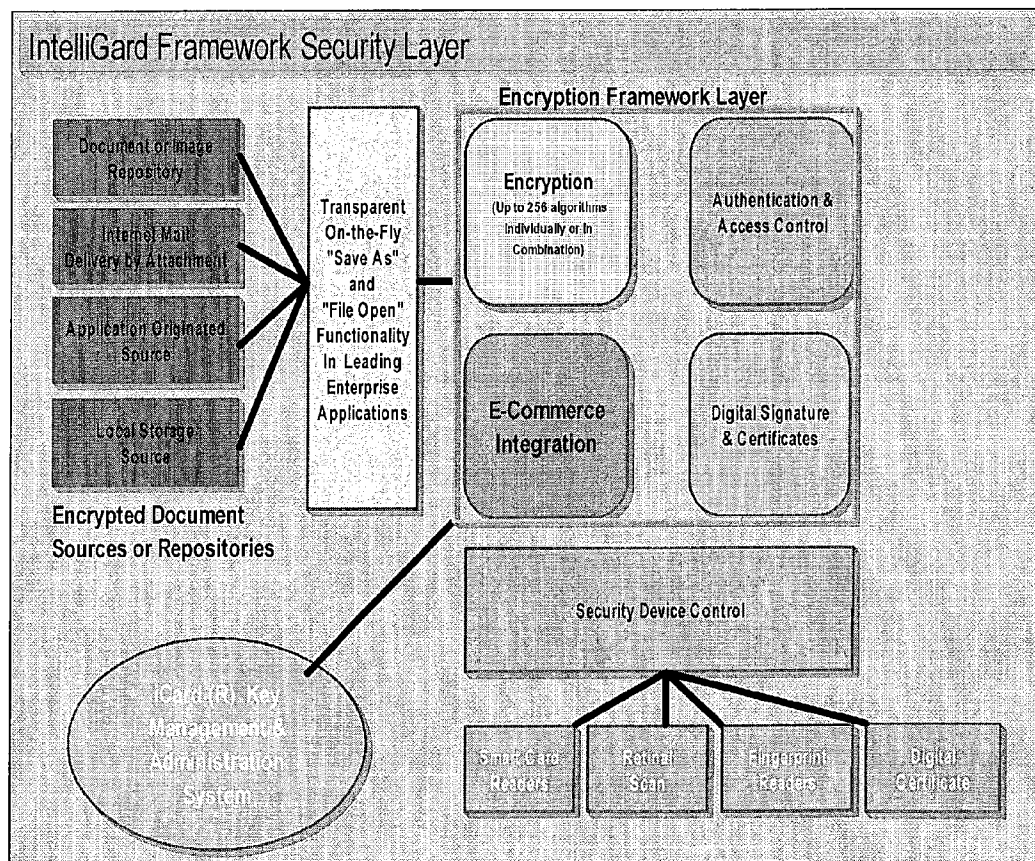
- **PUBLIC KEY ENCRYPTION**

Uses a pair of keys, one public and one private. Each user has a personal key pair, and the user's public key is used by other persons to send encrypted messages to the user, while the private key is employed by the user to decrypt messages received. Public key encryption algorithms include the public domain Diffie-Hellman algorithm, the RSA algorithm invented by Rivest, Shamir and Adleman at the Massachusetts Institute of Technology (MIT), Pretty Good Privacy algorithm (PGP) invented by Phil Zimmermann and ECC by Certicom. Because of their mathematical structure, public key encryption is slower than most private key systems, thus making them less efficient for use in a trusted network or for encrypting large files.



IntelliGard PC DOCS Enabled Document Security

IntelliGard brings a new dimension to today's corporate information security needs. IntelliGard is built around a secure cryptographic engine that supports up to 256 different encryption algorithms. The IntelliGard cryptographic interface can either be software based or enhanced by using state-of-the-art smart card technology to keep the encryption keys off-host and provide positive user identification. A user's smart card contains a unique user ID, password, and a user definable hierarchy of encryption keys. File sharing is accomplished by distributing a common key to members of a selected work group. The number of work groups that the user is a member of determines the number of keys on the smart card. This concept establishes a "need to know" hierarchy of access rights to company information. It also ensures simple key management. IntelliGard supports industry standard ISO 7816 compliant smart cards and smart card readers.



IntelliGard's open architecture design allows users the ability to custom design their document security system around their needs. IntelliGard works inside of PC DOCS to provide maximum document security and positive user identification with virtually zero disruption to their PC DOCS workflow. Custom configuration ensures rapid deployment. Minimal user training is required. IntelliGard makes encryption and decryption simple and easy to use.

IntelliGard PC DOCS Enabled Document Security Features

- Supports up to 256 different encryption algorithms (Exportable to anywhere in the World).
- Proprietary software based or smart card based key management system.
- Positive user identification using state of the art smart card technology.
- Trusted work group key distribution hierarchy system.
- Proprietary smart document file header for multiple encryption algorithms.
- Seamless file encryption and decryption interface.
- Multiple batch file encryption and decryption functions.
- Open architecture design provides rapid deployment and custom system configuration.
- Minimal user training.
- Fully integrated into the PC DOCS family of products


"Smart Security for the Future of Document Management."

EXHIBIT Z-21

IntelliGard for PC DOCS

Administrator Manual

Declaration of Stephen Zizzi



Introduction	2
Chapter 1: Installing IntelliGard	3
Hardware:	3
Operating System:	3
DOCS Open Users:	3
Serial Smart Card Reader Installation:	3
PCMCIA Smart Card Reader Installation:	4
Testing the smart card reader:	4
Figure 1-1: Test SwapSmart	4
Figure 2-1: Log in	6
Figure 2-2: Loading Keys	6
Figure 2-3: Change Password	6
Figure 2-4: Change User Password	7
Figure 2-5: Key Server	7
Right mouse click functions	8
Figure 3-1: Smart card functions	8
View Keys	8
Figure 3-2: IntelliGard Keys	8
Configure the File Names and System Access	9
Figure 3-4: Configure	9
Figure 3-5: Options	9
File Names	9
Figure 3-6: System Access	10
System Access	10
Halting the Server	11
Figure 3-7: Halt Server	11
Figure 3-8: Key Server	11
Starting the Server	11
Figure 4-1: DOCS Open Desktop	12
Create the MAZTABLE	Error! Bookmark not defined.
Figure 4-2: DOCS Open Designer Desktop	12
Figure 4-3: Select Field to Edit	13
Figure 4-3: Table Definition Dialog	13
Figure 4-4: Edit Column Description Dialog	14
Saving the MAZTABLE	15
Figure 5 DOCS Open Designer	15
IntelliGard Options inside the DOCS Open Desktop	16
Encrypt Batch	16
Decrypt Batch	16
Encrypt Selected	16
Decrypt Selected	16
Secure E-mail	16
Encrypt and Decrypt Batch	16
Batch Encrypt	16
Batch Decrypt	18
Figure 5-	18

Introduction

Welcome to IntelliGard

Today the need to exchange information through advanced technologies such as: LAN, WAN, Intranets, Extranets, Enterprise Document Management (EDM), and the Internet "secure" password protection is not enough. MAZ Technologies, Inc. has provided the framework for the exchange of information without altering the daily workflow.

IntelliGard is Windows based product that provides file level security for PC users. IntelliGard is simple easy to use and easy to learn.. A point and click Windows interface is provided allowing the user to encrypt and decrypt files on their PC, LAN, WAN, Intranet, Extranet, or Internet with ease and efficiency. With a simple right mouse click a single user or group of users may encrypt, decrypt and shred files. All files are encrypted using the Data Encryption Standard (DES) algorithm and an encryption key.

The encryption key comes from a secure; credit card size storage device called a smart card. IntelliGard makes use of smart cards to store and distribute encryption keys. The IntelliGard system requires a user name and password to run and grant access to the keys stored. If someone attempts to break into a smart card by guessing the password, the smart card is programmed to lock out further attacks after a predetermined number of tries. A smart card reader comes with IntelliGard System that can easily be connected to the user's PC.

In a multi-user environment, using numerous smart cards can be generated sharing a common set of keys. These smart cards can be distributed to a group of users who wish to exchange files securely. Those who do not have smart cards with an authorized set of keys will not be able to access protected files.

Chapter 1: Installing IntelliGard

System Requirements

The IntelliGard 2.1.0 installation CD-ROM contains installation software for the IntelliGard security software and hardware installation. The IntelliGard system is designed to run on the client side of a client/server network. The minimum system requirements are:

Hardware:

- IBM® compatible 486 PC
- 16 Mbytes RAM
- 3 Mbytes free hard disk space
- CD-ROM drive
- 1 SI/O serial port or 1 PCMCIA slot

Operating System:

Microsoft Windows® 95, 98, NT4.0

DOCS Open Users:

DOCS Open 3.7.x

Smart Card Reader Installation

Serial Smart Card Reader Installation:

The Serial Smart Card Reader has a single cable split into two connectors a 9 PIN DIN (DB-9) connector and a 6 PIN DIN keyboard tap.

1. Turn off your computer
2. Plug in the DB-9 smart card reader connector into an open serial port on the workstation.
3. Unplug the keyboard.
4. Plug in the remaining smart card reader connector into the keyboard port.
5. Plug the keyboard into the back of the smart card reader connector.
6. Start Windows
7. Insert the IntelliGard Installation CD-ROM.
8. Click on the Windows Start menu, and then click **Run**.
9. Open the IntelliGard CD-ROM
10. Open the Hardware folder
11. Open the Serial folder
12. Double Click "scm-setup.exe".
13. Follow the setup dialog to complete the hardware installation.

PCMCIA Smart Card Reader Installation:

Make sure that the Hardware is **NOT** inserted in the PCMCIA port. You will be asked to insert the PCMCIA Smart Card Reader during the hardware setup routine.

1. Start Windows
2. Insert the IntelliGard Installation CD-ROM.
3. Click on the Windows Start menu, and then click **Run**.
4. Open the IntelliGard CD-ROM
5. Open the Hardware Setup Folder
6. Open the Pcmcia folder
7. Double Click on the Setup.exe
8. Follow the setup dialog to complete the hardware installation.

It is advised that you test the smart card reader before installing the IntelliGard software.

Testing the smart card reader:

1. Start >CT-API SwapSmart SDK V1.14 > Test CT 95
2. Select "Test CT" from the CT-API menu in the program
3. A new dialog appears as shown in Figure 1-1

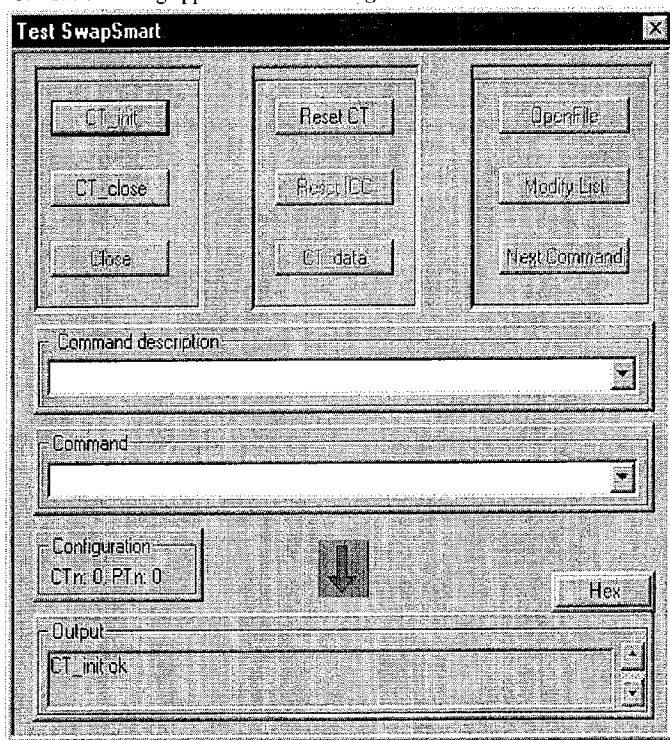


Figure 1-1: Test SwapSmart

4. Click CT_init
5. Check the Output window for CT_init OK. In addition a green arrow should appear above the Output window. (If a red arrow appears the Hardware is not installed correctly.)
6. You must click CT_close before continuing
7. You may now exit the test program.

IntelliGard software installation

1. Start Windows
2. Insert the IntelliGard CD in the CD-ROM drive. The installation program should launch automatically. If this does not occur follow these steps and the on-screen instructions.
3. In **Windows 95** and **NT 4.0**, click the Windows Start menu, and then click **RUN**
4. When the **Run** dialog box appears, type "x:\setup" Where x is the letter of your CD-ROM drive.
9. Click **OK**.
10. Reboot

Chapter 2 Logging In to the IntelliGard Smart Card

When you start Windows the IntelliGard Server will automatically start.

1. Insert your Smart card into the SCM Reader.
2. Enter the default password



Figure 2-1: Log in

3. IntelliGard will load the keys for your smart card as shown in figure 2-2.



Figure 2-2: Loading Keys

Change the default password to your personal password.

1. Right click on the IntelliGard Icon in the Taskbar tray.
2. Select Change Password.

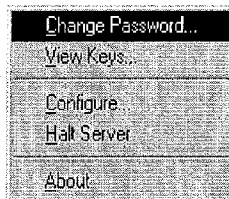


Figure 2-3: Change Password

The Change User Password dialog box appears (figure 2-4)

1. Enter the old password
2. Enter New Password
3. Confirm the new Password

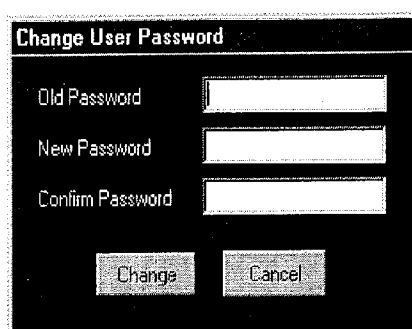


Figure 2-4: Change User Password

The KeyServer will confirm that your Password has been changed as shown below.

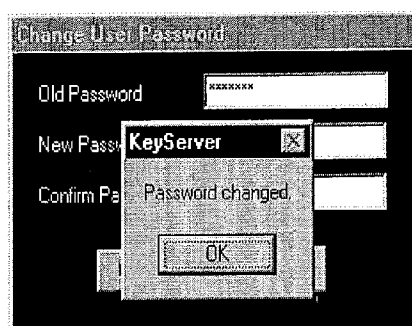


Figure 2-5: Key Server

Chapter 3 IntelliGard Administrator functions

Right mouse click functions

Right clicking on the icon in the taskbar tray provides you with several functions.

The Administrator Card allows you to Change Passwords, View Keys, Configure IntelliGard, Halt the Server, and About as show in the figure below.

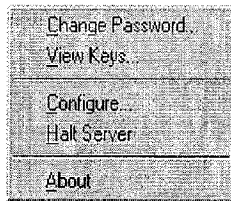


Figure 3-1: Smart card functions

View Keys

To view the keys available on your smart card follow these steps:

1. Right mouse click on the icon in the taskbar tray
2. Select View Keys
3. The IntelliGard Keys dialog displays as shown in figure 3-2

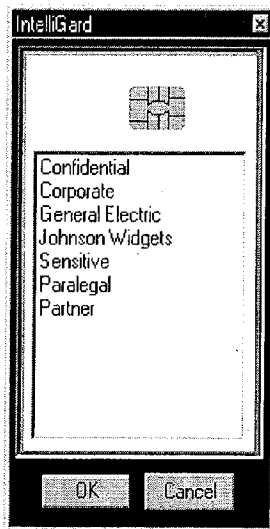


Figure 3-2: IntelliGard Keys

Configure the File Names and System Access

1. Locate the icon in the taskbar tray as shown in figure 3-3
2. Right click on the IntelliGard Icon



Figure 3-3: Taskbar Tray

3. Select Configure as shown in figure 3-4

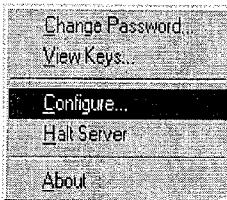


Figure 3-4: Configure

The Options dialog displays as shown below

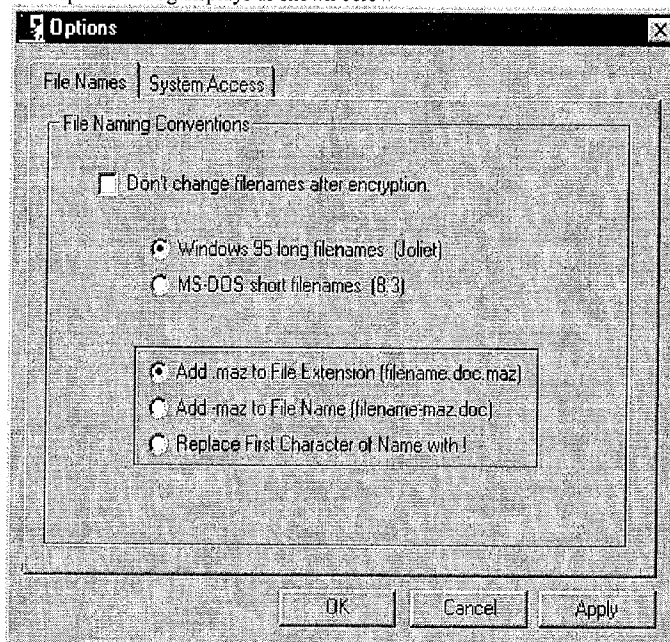


Figure 3-5: Options

File Names

The Options window defaults to the File Names Options. You may use Windows 95 long filenames or MS-DOS short file names. You may also change the way documents appear in the repository. If using DOCS Open it is not recommended that you change the default {Add .maz to File Extension (filename-maz.doc)}. Altering the .maz file extension could cause a failure in the decryption process inside DOCS Open.

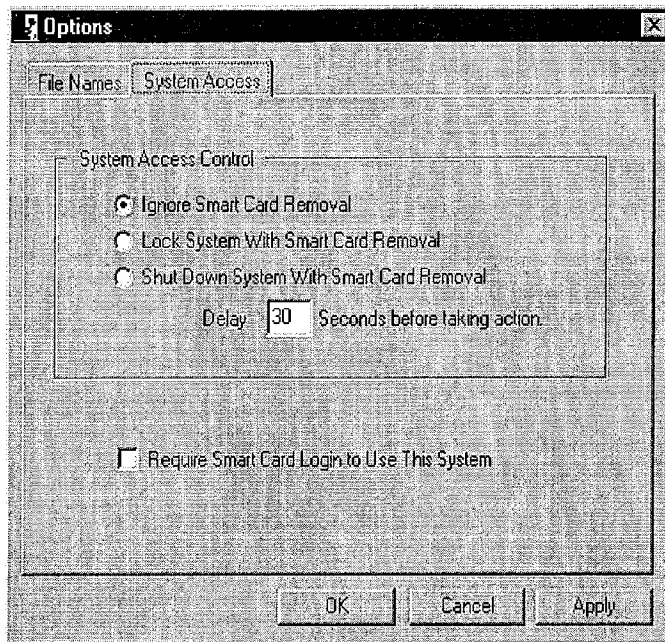


Figure 3-6: System Access

System Access

You have three choices for system access:

1. Minimum Security

Ignore Smart Card Removal: This option will allow you to remove your smart card without disruption to your workflow.

2. Medium Security:

Lock system with smart card removal: This option will start the IntelliGard Screen Saver. The smart card must be re-inserted into the smart card reader and logged-in in order to access the desktop.

3. Maximum Security.

Shut down system with smart card removal This option will close all programs and shut down your computer.

Enter the seconds to delay in the open box before taking action as shown in Figure 3-6.

*You should always require a Smart Card Login to use IntelliGard.

Halting the Server

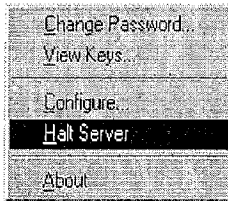


Figure 3-7: Halt Server

*This is a System Administrator function only and will not appear on a *User* smart card.

1. Right click on the IntelliGard Icon
2. Select **Halt Server**
3. Answer Yes to the question in figure 3-8

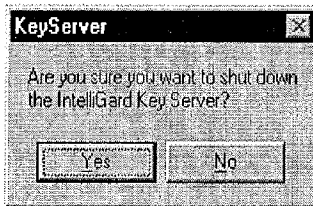


Figure 3-8: Key Server

Starting the Server

To start the key server once it has been halted access the IntelliGard Key Server:

Start > Programs > IntelliGard > IntelliGard Key Server

1. Log in to the IntelliGard Server as shown in figure 2-1

Chapter 4 Integrating with DOCS Open 3.7.X

The IntelliGard system integration with DOCS Open 3.7.x requires a table in the SQL database(s). The DOCS Open Designer Utility is used to create the new database table in SQL.

Creating the MAZTABLE

1. Launch the DOCS Open Desktop.

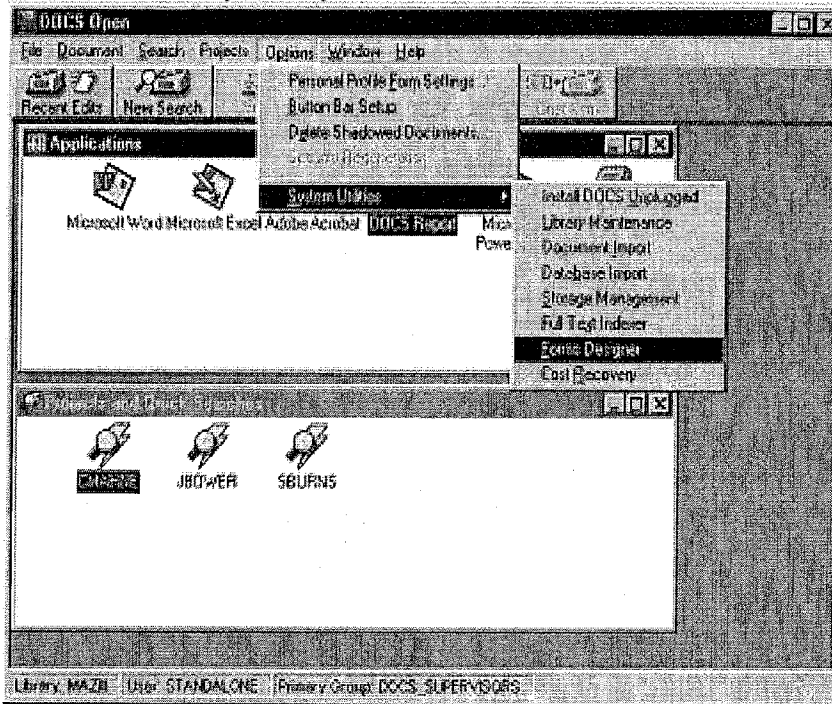


Figure 4-1: DOCS Open Desktop

2. Select Options
3. Select Forms Designer
4. Select the Database button.

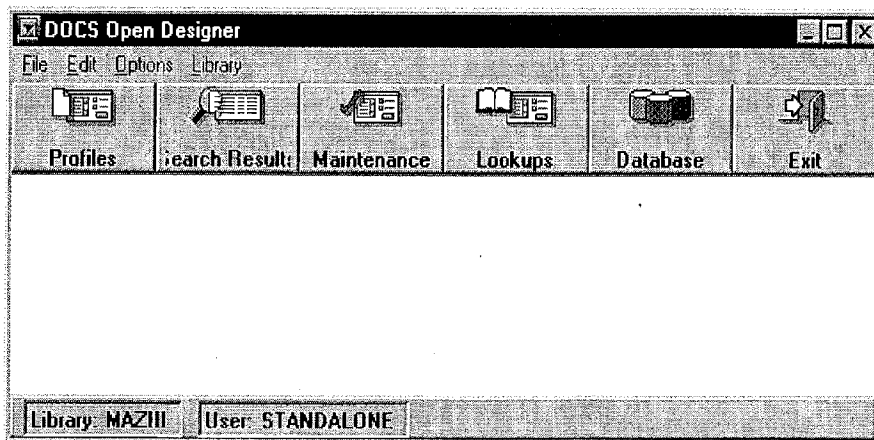


Figure 4-2: DOCS Open Designer Desktop

5. Click the New Table button

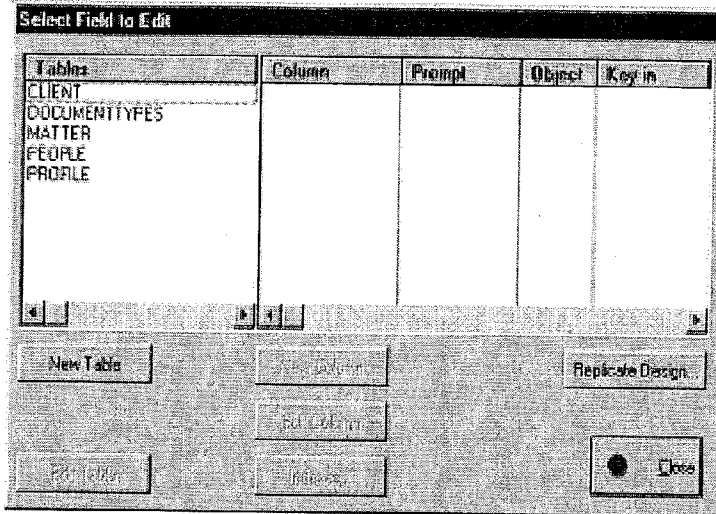


Figure 4-3: Select Field to Edit

Enter "MAZTABLE" in the Name field. (Figure4-3)

4. Enter IntelliGard Security in the Description field
5. Select OK

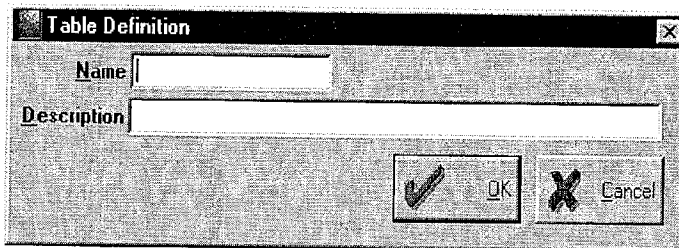


Figure 4-4: Table Definition Dialog

6. From the Table column, select the MAZTABLE

Figure 4-5: Edit Column Description Dialog

7. Define the following Columns:

Column Name: DOCNUMBER

Type: Integer

Key: Candidate

Object Type: Edit

Column Name: SECURE

Type: Integer

Key: No

Object Type: CheckBox

On Value 1

Off Value 0

Column Name: KEYID

Type: String

Key: No

Object Type: Edit

Length: 33

Saving the MAZTABLE

To save the completed table

1. Click on the Save Table button.
2. Create a new lookup called MAZTABLE?
3. Select **No** *The MAZTABLE is not used as a lookup

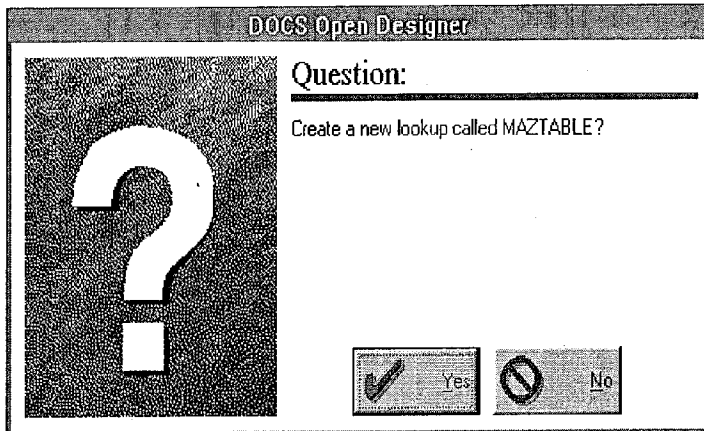
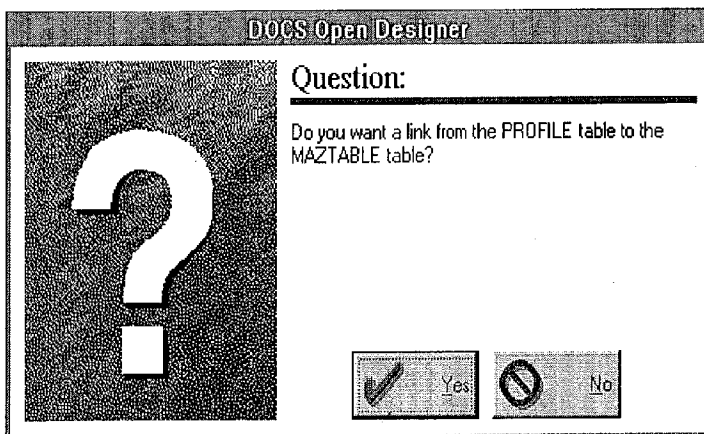


Figure 4-6: DOCS Open Designer

4. Create a Link



Chapter 5 Securing your files with the IntelliGard Encryption Interface

IntelliGard Options inside the DOCS Open Desktop

Encrypt Batch

Decrypt Batch

Encrypt Selected

Decrypt Selected

Secure E-mail

All of the following options are added to the DOCS Open desktop during installation. You may access these options from the "Document" pull down menu or to any Quick Search right mouse click pull down menu, documents in the DOCS Open file repository

Encrypt batch allows you to encrypt a group of files bases on criteria set by the DOCS Open profile information. For a DOCS Open "Default" library, selection criteria can be by author, by group, or by document type. If a "Financial", "Government", or "Legal" library is used, additional selection criteria are available to the user.

Encrypt and Decrypt Batch

From the DOCS Open Document

Selected and the Secure E-mail features are only available when a document or documents are selected in a DOCS Open search form.

Batch Encrypt

This feature was designed to secure mass amounts of files at one time with the same selection criteria. In order to use this function.

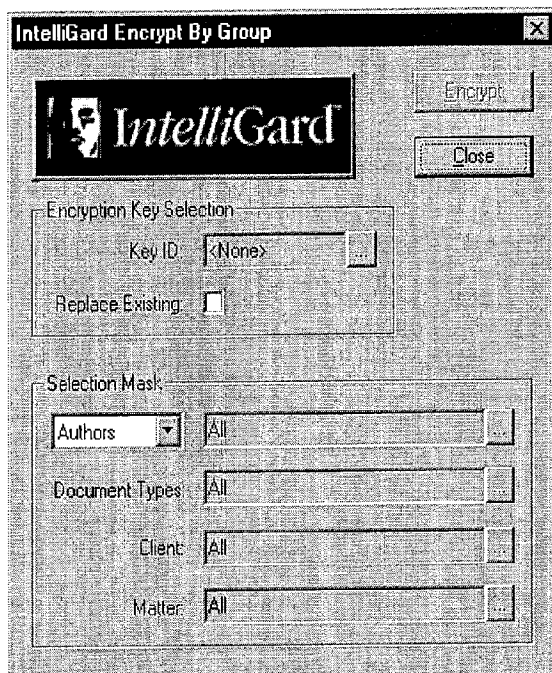


Figure 5-1

Follow these steps:

1. Select KEYID
2. Enter Password
3. Select Replace Exiting*
4. Choose Selection Mask either by Author or Group:
 - a) Group
 - b) Author
 - c) Document Type(Two additional selection masks are available when using a Legal Database)
 - d) Client
 - e) Matter
5. You may select more than one selection mask at a time.
6. Select ok
7. Select encrypt
8. Enter password
9. Select close

Batch Decrypt

This feature was designed to decrypt a group of files at one time with the same selection criteria.

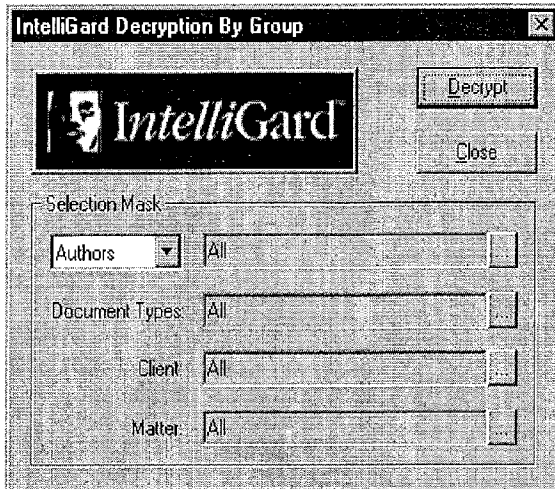


Figure 5-

Follow these steps:

1. Choose Selection Mask(s)* by:

- a) Group
- b) Author
- c) Document Type

(Two additional selection masks are available when using a Legal Database)

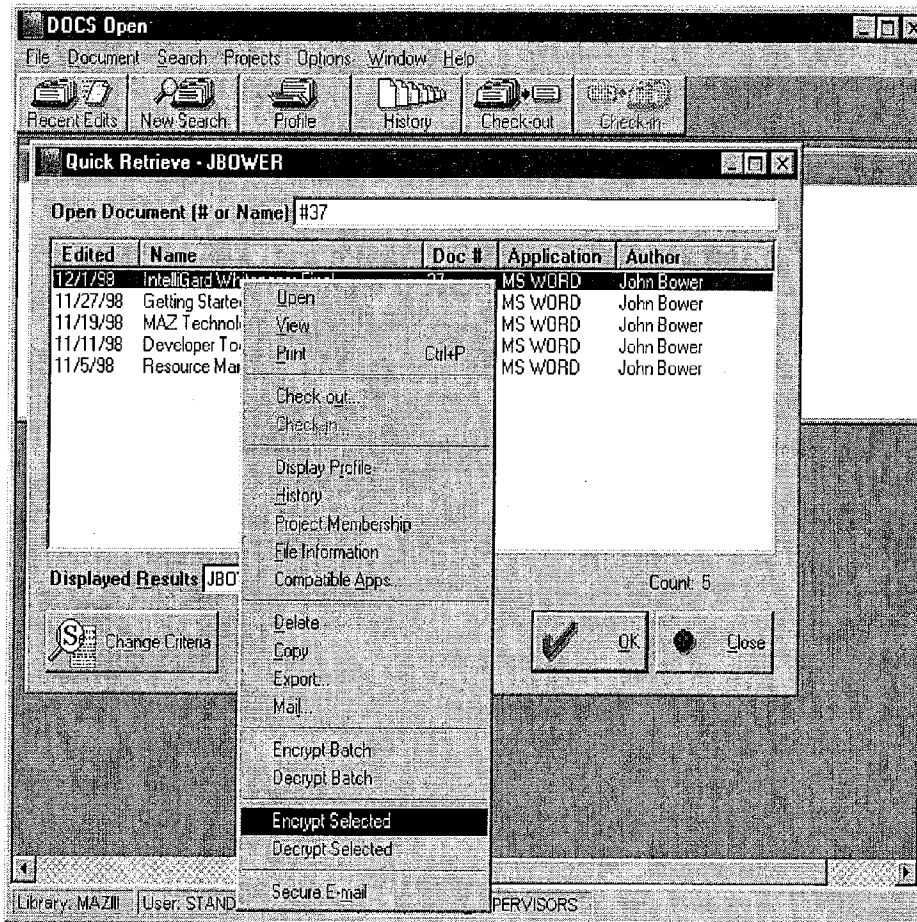
- d) Client
- e) Matter

*You may select more than one selection mask at a time.

2. Decrypt

3. Select Close

Encrypt Selected



Encrypt Selected:

Allows you to encrypt any file selected from a PC DOCS search form. To encrypt the file simply right mouse click on the selected file and the IntelliGard Encrypt Selected window will appear.

- 1.Right Mouse Click on Quick Search
- 2.Click on Encrypt Selected
- 3.Select Key ID
- 4.Enter password (not required for smart card users)
- 5.Click on Encrypt

Decrypt Selected:


Works exactly like encrypt selected with the exception that the key selection is not required.

- 1.Enter a password (not required for smart card users)
2. Click on Decrypt

EXHIBIT Z-22

Interim Report

Declaration of Stephen Zizzi



MAZ Technologies, Inc.

Specifications for MAZ Table generator.

Overview

The MAZ Table generator will be a DLL that can be called during program setup, or perhaps by an update or diagnostics program. The dll will connect to the database specified in the input parameters, and perform the functions required. The goal is to have a simple "one call" function that will **add the MAZ Technologies specific database table** to the DOCS Open database. The database can be any SQL database, not just MS SQL Server. The dll should not care if it is working on a local database, or a network database.

Timeframe

Naturally, the sooner the better. Please give us an idea how long this will take to create. The sooner the better. In an ideal world, we need the dll by **Monday the 7th**. In the real world, this may be different. Please let us know.

Language

The file should be a standard dll, preferably written in C++. VB requires too many support files to be used during the installation process. It practically requires a setup program itself.

DLL Location and Support Files Location

Any support files that the dll needs should be loaded from whatever the dll's location is. So, if it is copied to the file D:\temp\data\, it should look in that directory for all its support files.

If a specific file is to be referenced for support, it should accept the path to the file to use. That way if setup determines a different support file is necessary, the correct one can be sent in to the dll.

Functions

Required

A call to determine if a MAZ table exists in the current database specified.

A call to create a new MAZ table in the current database specified.

A call to remove a MAZ table from the database.

Future

A call to return data from a field in the table

A call to write data to a field in the table

A call to update an existing MAZ table to a new format.

Input Parameters

The dll should accept parameters specifying the location (path) of the database to which the table should be added. Other parameters as **recommended or required** for operation.

The dll should have all required data such as the data structure of the MAZ table in an external file to facilitate upgrading future revs. Perhaps a simple scripting language to add the table?

MAZ Table Generator, cont.

Synchronous Processing

The dll should not return from the call to add the table until the table has been successfully added, or a failure has occurred.

In a similar fashion, it should not return from any other call until it is finished.

(What I'm trying to avoid is launching an asynchronous process and having the setup program finish or reboot before the work is done.)

The dll should timeout and return an error if a problem occurs, to avoid hanging the installation process.

Progress Bar

If the process of adding the table will take a noticeable length of time, the function should display a progress bar.

The dll should accept a string value to display on the progress dialog, as well as a boolean value to display, or not display the progress dialog.

Return Values

The dll should return error codes representing the various problems that can occur during the process, so an intelligent response can be made to an error without aborting the install.

When practical, the return codes should be standard Microsoft error return constants. E.G. `ERR_SUCCESS`, etc.

The dll should return 0 on success

The dll should return an error if the table exists, unless it's the update function, of course.

Documentation

The dll should be accompanied by a simple text file documenting the API, error return codes, required support files and dependencies, and any other comments.

MAZ Table Generator, cont.

Additional Information

The following is an excerpt of a Readme file which instructs a user on the methods for adding the MAZ table manually. This is the specific process which we wish to automate.

Step III.

THIS STEP IS NOT NEEDED FOR USERS OF IMAIL. If you are using the full IntelliGard Keyserver installation, you must proceed with the following step:

After you have completed steps I. and II. you must build a New Table in the SQL database used with DOCS Open. Call this table MAZTABLE (very important!).

Open Designer(DOCs Forms Designer)

Select the Database Maintenance icon

Select New Table

Name the Table MAZTABLE

Description: "IntelliGard Security"

Select New Column to add the following three columns:

Column 1. "DOCNUMBER" type= edit, integer, candidate key

Column 2. "SECURE" type= checkbox, integer, on value = 1, off value = 0

Column 3. "KEYID" type = edit, string, length = 33

Save the new table.

Do not create a new lookup for MAZTABLE.

Create a join with the PROFILE table.

MAZ Technologies, Inc.

MAZ Technologies IntelliGard KeyServer Software Specification (Brief)

Overview

The following is a brief outline of the full software specification. The intention is to provide enough information to facilitate design and planning of the **CyberDocs integration**.

The core of IntelliGard smart card integration is the **KeyServer**. The KeyServer is an **ActiveX executable** which provides an interface which will allow a calling program to retrieve an encryption key from a user's smart card.

To obtain a key, the calling program simply needs to create an instance of the **server class, clsKeyServer**.

Two methods can be used to retrieve the key from the server, as outlined below.

The KeyServer handles all access to the card, as well as password changes, preference settings, and basic system lockouts. The lockout is merely intended as a first layer tactic to discourage prying. The primary function of the KeyServer is to provide keys obtained from the smart card.

File encryption is **not** performed by the KeyServer, with the exception of the "NotifyFileDaemon" method.

Architecture overview:

The IntelliGard system consists of several parts.

The **Proxy Server** is an ActiveX dll which receives **all calls to the server**. It then forwards the calls to the server if it is present, or to iMail as necessary for passphrase services.

The **Proxy Server can be queried** to determine if the real KeyServer is present on the user's machine. See the ValidRequest method below.

If the keyserver is not present, calls to the various keyserver methods will simply return undefined values. Before calls to the keyserver are made, **the proxy should be queried for the existence of the keyserver**.

The **KeyServer** is an ActiveX executable which services requests from the Proxy Server.

The KeyServer interfaces with the hardware through the reader engine.

The **ReaderEngine** is an ActiveX dll which interfaces directly with the actual hardware, or with the software smart card object.

The reader engine presents itself as an actual piece of hardware, so the server never knows if it is talking to a real smart card, or the **softcard**.

The **SoftCard** is a software structure that emulates a real smart card. It is "inserted" as a class in the ReaderEngine. The instance of the card interface inside the reader engine determines if calls are returned from the hardware object or the software object.

iMail is an ActiveX executable that provides passphrase encryption services through the server.

When a passphrase is requested from the server, the server forwards this request to iMail, then returns the result obtained.

Platforms: Win 95, 98, NT 4.0

Language: MSVB 5.0 SP3

Dependencies: Available on request.

KeyServer Properties:

NullKeyValue(ByVal szKeyName As String) As String

This property accepts a known key name and returns the value of the key as read from the smart card. This property has been maintained for interface compatibility. Better to use the function below.

KeyValue(szKeyName As String) As String

This function accepts a known key name and returns the value of the key as read from the smart card.

KeyServer Methods:

ValidRequest(code As String) As Boolean

Returns TRUE if the KeyServer is present. "Code" is disregarded. (This function was "dead code," which I was forced to keep for interface compatibility. I have re-used it to kludge in the Proxy Server query.)

DisplayReturnKey(szKeyName As String, szKeyValue As String) As Boolean

Displays the available keys to the user, allowing them to select a key to use for encryption. The key name and the key value are returned by reference through the calling variables. The method returns true if the returned data is valid.

NullDisplayReturnKey(szKeyName As String, szKeyValue As String) As Boolean

Calls the DisplayReturnKey method above, and appends an extra NULL character to the end of each string value. Don't ask me why.

NotifyFileDaemon(szFile As String, szKeyID As String) As Boolean

This method watches for the filename passed in to be created, then encrypts the file with the keyname passed in. This method is useful for encrypting a file that does not yet exist at the time the encryption call is made. This is a workaround for asynchronous file operations.

NullDisplayReturnECB(szKeyName As String, szKeyValue As String) As Boolean

This method calls iMail if it is installed, allowing the user to enter a passphrase on which a key will be generated. The generated key is then returned from iMail through this function.

Other Properties and Methods

Any properties or methods not outlined above are either unsupported or extraneous to the use of IntelliGard services.

There is a function that will return a filename massaged according to user preferences, which not be necessary with CyberDocs.

Typical use examples:

Obtaining a key name selected by the user, along with its value:

' It is advisable to instance the keyserver at the global level.

Dim KeyServer As New clsKeyServer

Dim szKeyName As String ' Holds the Name of the user - selected key

Dim szKeyVal As String ' Holds the value of the user- selected key

' Get a key from the server

KeyServer.DisplayReturnKey szKeyName, szKeyVal

Obtaining a passphrase from the user, and its generated value:

Dim szKeyName As String ' Holds the name of the key, for passphrase, always "CodeBook"

Dim szValue As String ' Holds the value generated from the passphrase.

KeyServer.NullDisplayReturnECB szKeyName, szValue

Scheduling a file to be encrypted by the server as soon as it is created.

Note that the server will timeout if the file does not appear within 60 seconds.

The boolean return from this function is meaningless.

' Pass the file name to the server, with the name of the key to use for encryption.

KeyServer.NotifyFileDaemon "c:\temp\file.txt", szKeyName

Additional Information

When a file is encrypted by the MAZ encryption engine, the name of the key which was used to encrypt the file is embedded in the header.

When the file is to be decrypted, the key name is retrieved from the header, then passed to the KeyServer (via the Proxy). The value of the key is obtained from the smart card by the KeyServer, then returned to the calling program.

The key value is then passed to the MAZ encryption engine for decryption of the file.

The **encryption engine does not talk to the server**. It is up to the implementation of the User Interface to obtain the key name and value from the server, then call the encryption engine for processing.

Typical steps to encrypt a file:

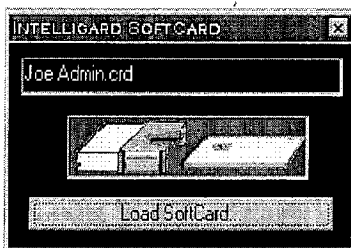
1. Obtain the target filename and path
2. Call the server to get a key and key value from the user
3. Call the encryption engine with the input filename, output filename, key name, and key value

Typical steps to decrypt a file:

1. Obtain the target filename and path
2. Call the encryption engine ("PrepDec") with the filename to prepare for decryption. PrepDec will return the name of the key that was used for encryption.
3. Call the Proxy Server with the name of the key, to obtain its value.
4. Call the encryption engine ("GetOriginalName") to get the original name of the encrypted file specified by the call to PrepDec. (If necessary.)
5. Call DecFile or DecFileNoDestroy with the input filename, output filename, and key value to use for the decryption.
6. Call PrepDec cleanup to cause the engine to clean up. I didn't write the engine.

Full specifications for the MAZ Encryption engine available upon request.

If the user is using SoftCards instead of actual smart cards for the encryption process, there will be no difference in the behavior of the server. However, a small GUI will appear, allowing the user to select the SoftCard they would like to "insert" into the reader. The SoftCard UI is shown below:



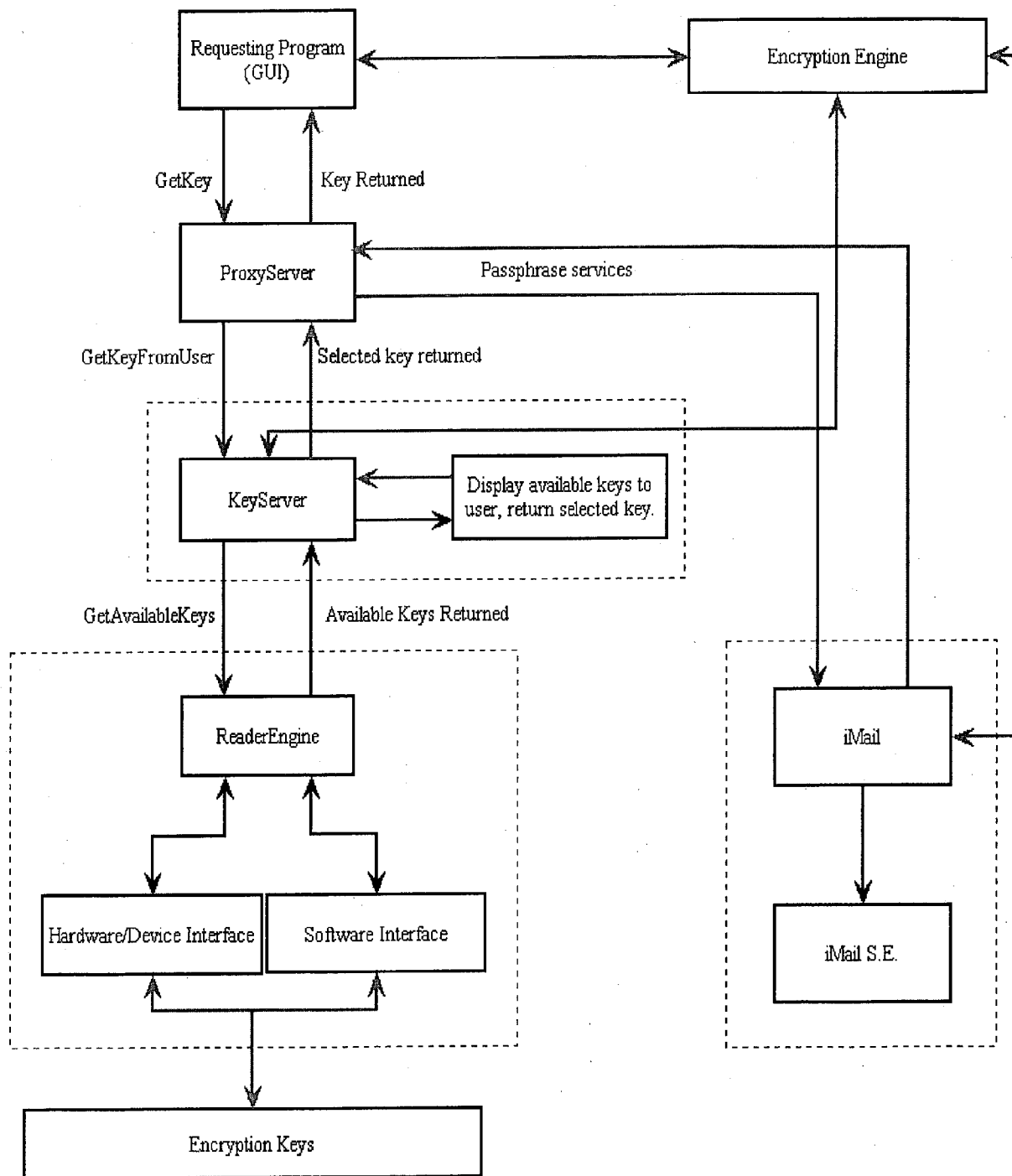
MAZ Technologies Source Code Resource and Design

Overview:

All MAZ Technologies software components take advantage of ActiveX/COM technology. The core components are the Encryption Engine, the Keyserver, iMail, the ProxyServer and the ReaderEngine. The ProxyServer routes ActiveX calls to and from iMail and the Keyserver to the requesting component. The Encryption Engine is driven directly by its API, from the GUI. The reader engine provides the interface between the Keyserver and the hardware. The IntelliGard softkey interface is presented by the Reader Engine.

In a typical call to the Keyserver, an ActiveX method such as "DisplayKeys" is called on the ProxyServer. The ProxyServer determines if the KeyServer is available. If it is, the ProxyServer passes the request to the KeyServer. The KeyServer calls the ReaderEngine for the list of keys on the smart card. The list is displayed by the KeyServer. If a user selects a key, the key is passed back through the ProxyServer to the original calling program.

This process is illustrated on the following page.



Defining program language choices.

Data encryption and decryption are microprocessor intensive processes. With this in mind, the core of all MAZ Technologies software is written in C and C++, a language known for speed. The Graphical User Interface (GUI) is typically written in Visual Basic, known for being a Rapid Application Development (RAD) programming language.

By leveraging these two languages for their best capabilities, product development can proceed at maximum speed with minimum performance impact.

The following components of MAZ Technologies products are listed with the language in which they are written.

DemoPad	VB
Encryption Engine	C++
IMail	VB
IMail SE	C
IMail Word Integration	VBA
IManage Integration	VB
IntelliGard Keyserver	VB
IWatch	C++
Key Diagnostics	VB
MAZ Console	VB
MAZ Interface	VB
PC DOCS integration	C++
ProxyServer	VB
ReaderEngine	VB
Registration Gen	VB
SmartCard Admin	VB
SmartCard Init	VB

In all cases, source code is written in classes to facilitate code re-use, with certain exceptions where execution speed and a minimal memory footprint are crucial.

All MAZ Technologies source code is written to the following specification:

MAZ Technologies Source Code Format Specification

1. Where ever possible, source must be written in classes to facilitate re-use.
2. Source code must be commented liberally and frequently, except in cases in which the execution of the code in question is obvious. In complex routines, comments should be on nearly every line, to explain the implementation as well as expected results. See fig. 1 in Appendix A for examples.
3. All source code must be formatted using the tab key, set to default at 4 spaces. Indentations will begin with the line after the opening statement in any logical structure. See fig. 2 in Appendix A for examples.
4. The end of any logical structure will be signified by the positioning of the closing statement at the same level as the opening statement. See fig. 2 in Appendix A for examples.
5. Constants will be used in all cases where appropriate, and shall be defined in a separate file unless the constant is part of a private class, such as in VB.
6. String tables, string constants, and resource files should be used in all cases where strings are designated, to facilitate changes and internationalization.
7. All temporary file operations must occur in the O.S. specified temporary path.
8. Program settings and user data must be stored in the registry, except where encryption of settings is mandated by the nature of the program being written.
9. All logical procedures should be broken down to a set of small functions.
10. All procedures should perform no more than one function.
11. Global variables must be avoided in all cases.
12. The "goto" command must not be used, except when forced to do so by the requirements of the keyword in use. E.g. the "error" keywords in VB.
13. Any looping statements or processor intensive functions should specifically yield to the O.S. during processing to avoid slow screen updates, which may disconcert the end user.
14. All coding architecture must take into consideration future expansion and changes.
15. All code must be "Y2K" compliant.
16. Foul language in source commentary is prohibited.
17. Language in any debugging output must be clear and professional. The end user must not be inadvertently presented with unprofessional messages. Ideally, such debug messages should be marked with a keyword comment to facilitate search and remove.
18. Error messages should not be cryptic. The user should be given some idea of a way to remedy the error that has occurred.
19. Design changes must be submitted for consideration before implementation.

Appendix A

Source code formatting examples.

Fig. 1 Source Code Comments

(This function can be found in iMail)

```
Private Sub GetIcon(szFilename As String)
' Gets the icon associated with the file contained in the archive.

    Dim hIcon As Long
    Dim szBuff As String
    Dim nRet As Long
    Dim szKeyName As String
    Dim szOrigName As String
    Dim i As ListImage

    ' Prep the file for decryption, so we can fetch the original filename
    szKeyName = Space$(255)
    nRet = PrepDec(szFilename, szKeyName)
    If nRet = EE_Success Then
        ' Get the original filename out of the header
        szOrigName = GetOutputFileName(szFilename)
    End If
    PrepDecCleanup

    ' Display the name of the original file in the listview of the main dialog
    frmCodeBook.lvwFiles.ListItems.Add , , GetFileName(szOrigName)
    ' Get the app associated with files of this type
    szBuff = GetAssociatedApp(szOrigName)

    If szBuff <> "" Then
        hIcon = ExtractAssociatedIcon(0, szBuff, 1)
        Set i = frmCodeBook.imglstImages.ListImages.Add(1, , CreateBitmapPicture(hIcon, 0,
vbPicTypeIcon))
    Else
        ' no icon found for this doc type
        Set i = frmCodeBook.imglstImages.ListImages.Add(1, ,
GetDefaultImage(GetFileExtension(szOrigName)))
    End If

    frmCodeBook.lvwFiles.SmallIcons = frmCodeBook.imglstImages
    frmCodeBook.lvwFiles.Icons = frmCodeBook.imglstImages
    frmCodeBook.lvwFiles.ListItems(1).SmallIcon = 1
End Sub
```

(Appendix A cont.)

Fig. 2 Source Code Indentation
(This function can be found in iMail)

In the following example, it can be seen that the logical block begins and ends with the Select/End Select block. Within the block, and indentation is used to indicate the beginning of a sub-block.

Public Function GetDefaultImage(szExt As String) As Image

' Returns the image to use for certain extensions

```
Select Case UCasc$(szExt)
  Case "DLL"
    Set GetDefaultImage = frmCodeBook.imgDll
  Case "EXE"
    Set GetDefaultImage = frmCodeBook.imgExe
  Case Else
    Set GetDefaultImage = frmCodeBook.imgDefault
End Select
```

End Function

In the next example, the logical block is defined by the For/Next structure.

```
For X = 1 To Len(szBuffer)
  a = Mid$(szBuffer, X, 1)
  Randomize Asc(a)
  n = Int(10 * Rnd)
  B = B & Trim$(Str$(n))
Next
```

Appendix A (cont.)

Source code formatting examples for C++
(Code taken from iWatch.cpp)

```
//
LRESULT IWatchWnd::OnTrayNotification(WPARAM wParam, LPARAM lParam)
{
    // Only handle commands with the same ID as icon tray
    if ((wParam != m_nid.uID) || (lParam != WM_RBUTTONDOWN))
        return 0;

    // Load the menu with the same ID as the ID of the icon data structure
    // Get the submenu of that for displaying off of the icon tray
    CMenu menu;
    if (!menu.LoadMenu(m_nid.uID))
        return 0;

    CMenu * pSubMenu = menu.GetSubMenu(0);
    if (!pSubMenu)
        return 0;

    if (lParam == WM_RBUTTONDOWN) {
        // Display the menu at the current mouse location. There's a "bug"
        // (Microsoft calls it a feature) in Windows 95 that requires calling
        // SetForegroundWindow. To find out more, search for Q135788 in MSDN.
        //
        int cmd;
        CPoint mouse;
        GetCursorPos(&mouse);

        ::SetForegroundWindow(m_nid.hWnd);
        if ((cmd = ::TrackPopupMenu(pSubMenu-
>m_hMenu, TPM_RETURNCMD, mouse.x, mouse.y, 0,
        m_nid.hWnd, NULL))) {
            PostMessage(WM_COMMAND, cmd, 0);
        }
    }

    return 1; // handled
}

//*****
//
void IWatchWnd::OnTrayExit()
{
    // send a close message
    SendMessage(WM_CLOSE, 0, 0L);
}

//*****
//
void IWatchWnd::OnDrawClipboard()
{
    CWnd::OnDrawClipboard();
}
```

```
// Get the clipboard owner.
m_pActiveWnd = GetForegroundWindow();

// Ignore the WM_DRAWCLIPBOARD sent by SetClipboardViewer()
if (!lg_bInCreate) {
    // Check if the clipboard data is the same as it was before
    CompareClipboardData();
}
// If there is another window in the chain send it the WM_DRAWCLIPBOARD msg
if (m_hNextWndChain)
    ::SendMessage(m_hNextWndChain, WM_DRAWCLIPBOARD, 0, 0);
}
```


MAZ Technologies Products and Status

IntelliGard KeyServer for DOCS Open 3.71

Support for smart card based file level encryption and decryption, as well as PC locking and security functions.

IntelliGard SoftCard technology emulates smart card functionality for powerful file encryption on a small budget.

Final release will have automatic integration during setup. However, Users will still need to create the MAZ table in the DOCS database.

Includes IntelliGard iMail for DOCS Open 3.71

Limitations:

Automatic full text indexing not supported.

"Save Version As" requires slight, easy workaround to use encryption.

IntelliGard iMail for DOCS Open 3.71

Supports pass phrase encryption of single files to be e-mailed from within DOCS Open 3.71. Automatically integrates with DOCS during setup.

Full version supports drag-and drop text encryption, multiple file encryption and decryption, full drag-and-drop file encryption, and single compressed, encrypted archiving functions.

Full version not yet available, missing many little bits.

Status

Besides the infamous icell bug, Clive needs to fix a simple bug that prevents Secure E-mail from showing up on the DOCS menus.

IntelliGard iMail Lite for clients

Supports single file pass phrase encryption and decryption for simple secure file exchanges with clients.

Status

Still has "icell" bug. Right now, it works most of the time, but it still fails too often. As soon as icell is fixed, it will be fully available.

IntelliGard iManage Smart Card Manager

Used to create keys for smart cards to be used with IntelliGard KeyServer. Also creates SoftCards for IntelliGard Softcard technology.

Status

Fully functional support for smart card creation, but does not yet fully support softcard creation yet. Also requires some work before the program security is fully activated.

Has no key escrow or key recovery yet. This is very important to our plan. Also needs to create configuration file and simple key install program which will allow an Administrator to customize how each PC functions with the IntelliGard KeyServer.

IntelliGard iWatch

Supports encapsulation of encrypted text in an OLE object. Also enables in-line encryption of text.

Status

Integrated into the IntelliGard line of products.

Summary:

When the icell bug is fixed, and the setup program is tweaked a little bit, the IntelliGard Keyserver for Smart Cards, iMail Lite for DOCS, and iMail Lite for Clients will be fully finished and releasable.

All other products require additional time before release. IWatch, and the iMail full versions are the next closest to being completed.

iManage requires the most work as of yet.

We also need two simple utilities for automating card formatting. This will enable us to quickly and easily create blank cards for use with iManage. I've got them nearly done, too, but they are on the back burner to everything else, because we can create the cards manually for now.

On the horizon: I think we have a huge opportunity with an encryption product that works exactly like WinZip, except the file is fully encrypted as well as compressed. We may even be able to work a deal with Niko Mak computing to create a plug-in for WinZip. Clive did some work with the guy who wrote WinZip, so we might have an edge there.

Right now, WinZip has password protection, but it's fairly easy to crack, like Word passwords. Imagine WinZip with IntelliGard encryption?

Who's working on what:

Clive and I are working on the iCell bug.

Clive is working on the iMail Lite for DOCS menu bug. (See above.)

Clive needs to make a change for the hardware support.

Edd Gross is working on iWatch

Shannon is working on help files

Obsidian will be working on iManage

I am working on everything else.

What we have right NOW:

Right now today we could give someone a working version of the IntelliGard Keyserver with setup support for both the serial and PCMCIA hardware devices.

It is not an ideal setup for the hardware, but it will work.

It also does not integrate automatically with DOCS yet. (I just haven't gotten to it yet.)

We can give people a version of iMail Lite for DOCS, and iMail Lite for Clients that will work most of the time, except for the ubiquitous icell bug.

iMail Lite for DOCS does integrate automatically with PC DOCS.

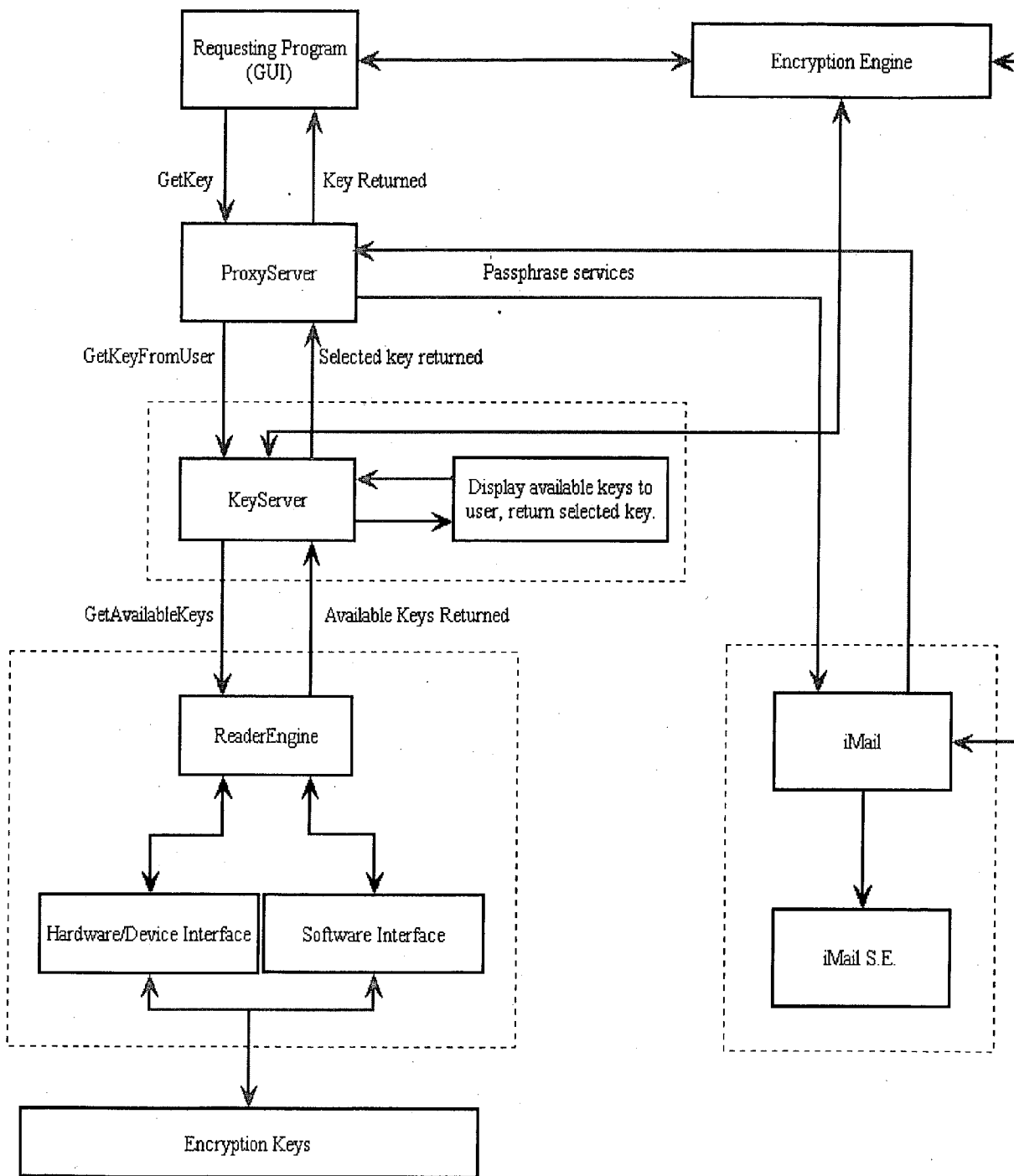
MAZ Technologies Source Code Resource and Design

Overview:

All MAZ Technologies software components take advantage of ActiveX/COM technology. The core components are the Encryption Engine, the Keyserver, iMail, the ProxyServer and the ReaderEngine. The ProxyServer routes ActiveX calls to and from iMail and the Keyserver to the requesting component. The Encryption Engine is driven directly by its API, from the GUI. The reader engine provides the interface between the Keyserver and the hardware. The IntelliGard softkey interface is presented by the Reader Engine.

In a typical call to the Keyserver, an ActiveX method such as "DisplayKeys" is called on the ProxyServer. The ProxyServer determines if the KeyServer is available. If it is, the ProxyServer passes the request to the KeyServer. The KeyServer calls the ReaderEngine for the list of keys on the smart card. The list is displayed by the KeyServer. If a user selects a key, the key is passed back through the ProxyServer to the original calling program.

This process is illustrated on the following page.



Defining program language choices.

Data encryption and decryption are microprocessor intensive processes. With this in mind, the core of all MAZ Technologies software is written in C and C++, a language known for speed. The Graphical User Interface (GUI) is typically written in Visual Basic, known for being a Rapid Application Development (RAD) programming language.

By leveraging these two languages for their best capabilities, product development can proceed at maximum speed with minimum performance impact.

The following components of MAZ Technologies products are listed with the language in which they are written.

DemoPad	VB
Encryption Engine	C++
IMail	VB
IMail SE	C
IMail Word Integration	VBA
IManage Integration	VB
IntelliGard Keyserver	VB
IWatch	C++
Key Diagnostics	VB
MAZ Console	VB
MAZ Interface	VB
PC DOCS integration	C++
ProxyServer	VB
ReaderEngine	VB
Registration Gen	VB
SmartCard Admin	VB
SmartCard Init	VB

In all cases, source code is written in classes to facilitate code re-use, with certain exceptions where execution speed and a minimal memory footprint are crucial.

All MAZ Technologies source code is written to the following specification:

MAZ Technologies Source Code Format Specification

1. Where ever possible, source must be written in classes to facilitate re-use.
2. Source code must be commented liberally and frequently, except in cases in which the execution of the code in question is obvious. In complex routines, comments should be on nearly every line, to explain the implementation as well as expected results. See fig. 1 in Appendix A for examples.
3. All source code must be formatted using the tab key, set to default at 4 spaces. Indentations will begin with the line after the opening statement in any logical structure. See fig. 2 in Appendix A for examples.
4. The end of any logical structure will be signified by the positioning of the closing statement at the same level as the opening statement. See fig. 2 in Appendix A for examples.
5. Constants will be used in all cases where appropriate, and shall be defined in a separate file unless the constant is part of a private class, such as in VB.
6. String tables, string constants, and resource files should be used in all cases where strings are designated, to facilitate changes and internationalization.
7. All temporary file operations must occur in the O.S. specified temporary path.
8. Program settings and user data must be stored in the registry, except where encryption of settings is mandated by the nature of the program being written.
9. All logical procedures should be broken down to a set of small functions.
10. All procedures should perform no more than one function.
11. Global variables must be avoided in all cases.
12. The "goto" command must not be used, except when forced to do so by the requirements of the keyword in use. E.g. the "error" keywords in VB.
13. Any looping statements or processor intensive functions should specifically yield to the O.S. during processing to avoid slow screen updates, which may disconcert the end user.
14. All coding architecture must take into consideration future expansion and changes.
15. All code must be "Y2K" compliant.
16. Foul language in source commentary is prohibited.
17. Language in any debugging output must be clear and professional. The end user must not be inadvertently presented with unprofessional messages. Ideally, such debug messages should be marked with a keyword comment to facilitate search and remove.
18. Error messages should not be cryptic. The user should be given some idea of a way to remedy the error that has occurred.
19. Design changes must be submitted for consideration before implementation.

Appendix A

Source code formatting examples.

Fig. 1 Source Code Comments
(This function can be found in iMail)

```
Private Sub GetIcon(szFilename As String)
' Gets the icon associated with the file contained in the archive.

    Dim hIcon As Long
    Dim szBuff As String
    Dim nRet As Long
    Dim szKeyName As String
    Dim szOrigName As String
    Dim i As ListImage

    ' Prep the file for decryption, so we can fetch the original filename
    szKeyName = Space$(255)
    nRet = PrepDec(szFilename, szKeyName)
    If nRet = EE_Success Then
        ' Get the original filename out of the header
        szOrigName = GetOutputFileName(szFilename)
    End If
    PrepDecCleanup

    ' Display the name of the original file in the listview of the main dialog
    frmCodeBook.lvwFiles.ListItems.Add , , GetFileName(szOrigName)
    ' Get the app associated with files of this type
    szBuff = GetAssociatedApp(szOrigName)

    If szBuff <> "" Then
        hIcon = ExtractAssociatedIcon(0, szBuff, 1)
        Set i = frmCodeBook.imglstImages.ListImages.Add(1, , CreateBitmapPicture(hIcon, 0,
vbPicTypeIcon))
    Else
        ' no icon found for this doc type
        Set i = frmCodeBook.imglstImages.ListImages.Add(1, ,
GetDefaultImage(GetFileExtension(szOrigName)))
    End If

    frmCodeBook.lvwFiles.SmallIcons = frmCodeBook.imglstImages
    frmCodeBook.lvwFiles.Icons = frmCodeBook.imglstImages
    frmCodeBook.lvwFiles.ListItems(1).SmallIcon = 1

End Sub
```


(Appendix A cont.)

Fig. 2 Source Code Indentation

(This function can be found in iMail)

In the following example, it can be seen that the logical block begins and ends with the Select/End Select block. Within the block, and indentation is used to indicate the beginning of a sub-block.

Public Function GetDefaultImage(szExt As String) As Image

' Returns the image to use for certain extensions

Select Case UCase\$(szExt)

Case "DLL"

Set GetDefaultImage = frmCodeBook.imgDll

Case "EXE"

Set GetDefaultImage = frmCodeBook.imgExe

Case Else

Set GetDefaultImage = frmCodeBook.imgDefault

End Select

End Function

In the next example, the logical block is defined by the For/Next structure.

For X = 1 To Len(szBuffer)

a = Mid\$(szBuffer, X, 1)

Randomize Asc(a)

n = Int(10 * Rnd)

B = B & Trim\$(Str\$(n))

Next

Appendix A (cont.)

Source code formatting examples for C++
(Code taken from iWatch.cpp)

```
//
LRESULT IWatchWnd::OnTrayNotification(WPARAM wParam, LPARAM lParam)
{
    // Only handle commands with the same ID as icon tray
    if ((wParam != m_nid.uID) || (lParam != WM_RBUTTONUP))
        return 0;

    // Load the menu with the same ID as the ID of the icon data structure
    // Get the submenu of that for displaying off of the icon tray
    CMenu menu;
    if (!menu.LoadMenu(m_nid.uID))
        return 0;

    CMenu * pSubMenu = menu.GetSubMenu(0);
    if (!pSubMenu)
        return 0;

    if (lParam == WM_RBUTTONUP) {
        // Display the menu at the current mouse location. There's a "bug"
        // (Microsoft calls it a feature) in Windows 95 that requires calling
        // SetForegroundWindow. To find out more, search for Q135788 in MSDN.
        //
        int cmd;
        CPoint mouse;
        GetCursorPos(&mouse);

        ::SetForegroundWindow(m_nid.hWnd);
        if ((cmd = ::TrackPopupMenu(pSubMenu-
>m_hMenu, TPM_RETURNCMD, mouse.x, mouse.y, 0,
        m_nid.hWnd, NULL))) {
            PostMessage(WM_COMMAND, cmd, 0);
        }
    }

    return 1; // handled
}

//*****
//
void IWatchWnd::OnTrayExit()
{
    // send a close message
    SendMessage(WM_CLOSE, 0, 0L);
}

//*****
//
void IWatchWnd::OnDrawClipboard()
{
    CWnd::OnDrawClipboard();
}
```

```
// Get the clipboard owner.
m_pActiveWnd = GetForegroundWindow();


// Ignore the WM_DRAWCLIPBOARD sent by SetClipboardViewer()
if (!g_bInCreate) {
    // Check if the clipboard data is the same as it was before
    CompareClipboardData();
}
// If there is another window in the chain send it the WM_DRAWCLIPBOARD msg
if (m_hNextWndChain)
    ::SendMessage(m_hNextWndChain, WM_DRAWCLIPBOARD, 0, 0);
}
```

EXHIBIT Z-23

IntelliGard Database

Query Program

Declaration of Stephen Zizzi



SQL VIEW

PC DOCS.

Option Explicit

Private Sub Command1_Click()

```

    Dim rc As Long
    Dim hSQLView As Long
    'Declare my variables
    Dim Library As String, userid As String
    Dim docNumber As String, DocName As String
    'Dim Text As String
    Dim docNum As Integer
    'Dim lpCaption As String
    'Dim NoUserID As Integer
    Dim LastEditDate As String
    Dim LastEditTime As String
    Dim Version_id As String
    'Dim EditDate As String
    'Dim EditTime As String
    Dim LastDateTime As String
    Dim LastVersion_ID As String
    Dim EditDateTime As String
    Dim Path As String

```

'Initialize my variables

```

Path = String(144, Chr(0))
Version_id = String(144, Chr(160))
LastEditDate = String(144, Chr(0))
LastEditTime = String(144, Chr(0))
Library = String(144, Chr(0))
userid = String(144, Chr(0))
docNumber = String(144, Chr(160))
DocName = String(144, Chr(160))
'lpCaption = "PC DOCS Open API Sample App"
'NoUserID = False
docNum = InputBox("Enter docnumber")
'Get the Library and UserID off dialog box
Library = "New-York"

```

'Get a new SQL View object from DOCS

hSQLView = DOCSSQLNewView(ByVal Library, ByVal "DOCSADM.Versions")

'Add columns to our SQL View that we want values back for...

```

rc = DOCSSQLAddColumn(hSQLView, ByVal "Version_ID", COLTYPE_INT)
rc = DOCSSQLAddColumn(hSQLView, ByVal "LasteditDate", COLTYPE_DATE)
rc = DOCSSQLAddColumn(hSQLView, ByVal "Lastedittime", COLTYPE_TIME)
rc = DOCSSQLAddColumn(hSQLView, ByVal "docnumber", COLTYPE_INT)

```

rc = DOCSSQLSetColumnValue(hSQLView, ByVal "docnumber", docNum)

'Issue the above Sql View to DOCS to run it for us...

rc = DOCSSQLSelect(hSQLView)

If rc = 0 Then

MsgBox "Either Select Failed Or Returned no data!", API_STOP

End If

'Do this loop while DOCSSQLNextRow keeps not returning 0 (more rows) do this...

While rc <> 0

'Get the DOCNUMBER and DOCNAME from DOCS...

rc = DOCSSQLGetColumnValue(hSQLView, ByVal "Version_id", ByVal Version_id, 144)

rc = DOCSSQLGetColumnValue(hSQLView, ByVal "docnumber", ByVal docNumber, 144)

rc = DOCSSQLGetColumnValue(hSQLView, ByVal "lasteditdate", ByVal LastEditDate, 144)

rc = DOCSSQLGetColumnValue(hSQLView, ByVal "lastedittime", ByVal LastEditTime, 144)

```

    EditDateTime = DateValue(Format(LastEditDate, "general date")) + TimeValue(Format(LastEditTime, "general date"))

```

If EditDateTime > LastDateTime Then

LastDateTime = EditDateTime

LastVersion_ID = Version_id

End If

```
'Get the Next Row from DOCS (move the cursor to the next logical row)
rc = DOCSSQLNextRow(hSQLView)
```

```
Wend
```

```
List1.AddItem "the version_ID " + LastVersion_ID + " is the latest version"
```

```
'Free the result set from memory (both server and client)
```

```
rc = DOCSSQLFreeResultSet(hSQLView)
```

```
'Free the View from DOCS memory
```

```
rc = DOCSSQLFreeView(hSQLView)
```

```
'hSQLView = 0
```

```
hSQLView = DOCSSQLNewView(ByVal Library, ByVal "DOCSADM.Components")
```

```
'Add columns to our SQL View that we want values back for...
```

```
rc = DOCSSQLAddColumn(hSQLView, ByVal "Version_ID", COLTYPE_INT)
```

```
rc = DOCSSQLAddColumn(hSQLView, ByVal "Path", COLTYPE_STRING)
```

```
rc = DOCSSQLSetColumnValue(hSQLView, ByVal "version_id", LastVersion_ID)
```

```
'Issue the above Sql View to DOCS to run it for us...
```

```
rc = DOCSSQLSelect(hSQLView)
```

```
If rc = 0 Then
```

```
    MsgBox "Either Select Failed Or Returned no data!", API_STOP
```

```
End If
```

```
'Do this loop while DOCSSQLNextRow keeps not returning 0 (more rows) do this...
```

```
'While rc <> 0
```

```
    rc = DOCSSQLGetColumnValue(hSQLView, ByVal "Version_id", ByVal Version_id, 144)
```

```
    rc = DOCSSQLGetColumnValue(hSQLView, ByVal "Path", ByVal Path, 144)
```

```
'Wend
```

```
List1.AddItem "The path is " & Path
```

```
'Free the result set from memory (both server and client)
```

```
rc = DOCSSQLFreeResultSet(hSQLView)
```

```
'Free the View from DOCS memory
```

```
rc = DOCSSQLFreeView(hSQLView)
```

```
End Sub
```

#If Win32 Then

'PC DOCS API Function Prototypes

' DMS Interface

Declare Function DOCSAddDocumentComponent Lib "docsap32.dll" (ByVal APPID As String, ByVal FileIn As String, ByVal componentPath As String) As Integer

Declare Function DOCSCloseDocument Lib "docsap32.dll" (ByVal APPID As String, ByVal fileName As String) As Integer

Declare Function DOCSOpenDocument Lib "docsap32.dll" (ByVal APPID As String, ByVal fileName As String) As Integer

Declare Function DOCSOpenDocumentReadOnly Lib "docsap32.dll" (ByVal APPID As String, ByVal fileName As String) As Integer

Declare Function DOCSOpenSelectedDocument Lib "docsap32.dll" (ByVal docNum As String, ByVal libName As String, ByVal APPID As String, ByVal docPath As String) As Integer

Declare Function DOCSQBEOpenDocument Lib "docsap32.dll" (ByVal criteria As String, ByVal APPID As String, ByVal openFileID As String) As Integer

Declare Function DOCSQBESelectDocument Lib "docsap32.dll" (ByVal criteria As String, ByVal docNum As String, ByVal libName As String) As Integer

Declare Function DOCSReleaseReadOnlyDocument Lib "docsap32.dll" (ByVal APPID As String, ByVal fileName As String) As Integer

Declare Function DOCSSaveAsAttachment Lib "docsap32.dll" (ByVal APPID As String, ByVal openFileID As String, ByVal flags As Integer, ByVal versionLabel As String, ByVal docPath As String, ByVal shadowPath As String) As Integer

Declare Function DOCSSaveDocAsExt Lib "docsap32.dll" (ByVal APPID As String, ByVal fileName As String, ByVal flags As Integer, ByVal docPath As String, ByVal shadow As String) As Integer

Declare Function DOCSSaveDocument Lib "docsap32.dll" (ByVal APPID As String, ByVal fileName As String, ByVal docPath As String, ByVal shadow As String) As Integer

Declare Function DOCSSaveDocumentAs Lib "docsap32.dll" (ByVal APPID As String, ByVal fileName As String, ByVal docPath As String, ByVal shadow As String) As Integer

Declare Function DOCSSelectDocument Lib "docsap32.dll" (ByVal docNum As String, ByVal libName As String) As Integer

Declare Function DOCSSaveDoc Lib "docsap32.dll" (ByVal APPID As String, ByVal fileName As String, ByVal flag As Integer, ByVal docPath As String, ByVal shadow As String) As Integer

Declare Function DOCSSaveDocAs Lib "docsap32.dll" (ByVal APPID As String, ByVal fileName As String, ByVal flag As Integer, ByVal docPath As String, ByVal shadow As String) As Integer

'New or updated FUNCTIONS

Declare Function DOCSOpenSelectedDoc Lib "docsap32.dll" (ByVal docNum As String, ByVal libName As String, ByVal APPID As String, ByVal flags As Integer, ByVal docPath As String) As Integer

Declare Function DOCSOpenSelectedVersion Lib "docsap32.dll" (ByVal docNum As String, ByVal libName As String, ByVal verslabel As String, ByVal APPID As String, ByVal flags As Integer, ByVal docPath As String) As Integer

Declare Function DOCSAssignFileACLFromDB Lib "docsap32.dll" (ByVal APPID As String, ByVal fileName As String, ByVal readDB As Integer) As Integer

Declare Function DOCSQBESearch Lib "docsap32.dll" (ByVal docNum As String, ByVal docPath As String) As Integer

Declare Function DOCSSaveAsNewSubVersion Lib "docsap32.dll" (ByVal APPID As String, ByVal fileName As String, ByVal comments As String, ByVal Author As String, ByVal Typist As String, ByVal billable As String, ByVal ShowForm As Integer, ByVal docPath As String, ByVal shadow As String) As Integer

Declare Function DOCSSaveAsNewVersion Lib "docsap32.dll" (ByVal APPID As String, ByVal fileName As String, ByVal comments As String, ByVal Author As String, ByVal Typist As String, ByVal billable As String, ByVal ShowForm As Integer, ByVal docPath As String, ByVal shadow As String) As Integer

Declare Function DOCSDeleteDocument Lib "docsap32.dll" (ByVal docNum As String, ByVal libName As String, ByVal flags As Integer) As Integer

Declare Function DOCSAddRouteActivity Lib "docsap32.dll" (ByVal docNum As Long, ByVal libName As String) As Integer

Declare Function DOCSSelectProfileForm Lib "docsap32.dll" (ByVal libName As String, ByVal APPID As String, ByVal flags As Integer) As Integer

Declare Function DOCSSelectQBForm Lib "docsap32.dll" (ByVal flags As Integer) As Integer

Declare Function DOCSEnableErrorMessages Lib "docsap32.dll" (ByVal flags As Integer) As Integer

Declare Function DOCSRemoveSecurity Lib "docsap32.dll" (ByVal docNumber As String, ByVal Library As String) As Integer

Declare Function DOCSAddTrustee Lib "docsap32.dll" (ByVal trustee As String, ByVal docNum As String, ByVal libName As String, ByVal rights As Integer, ByVal flags As Integer) As Integer

Declare Function DOCSRemoveTrustee Lib "docsap32.dll" (ByVal trustee As String, ByVal docNum As String, ByVal libName As String, ByVal flags As Integer) As Integer

Declare Function DOCSNewTrusteeHandle Lib "docsap32.dll" (ByVal docNum As String, ByVal libName As String) As Integer

```

Declare Function DOCSNextTrustee Lib "docsap32.dll" (ByVal htrustee As Integer, ByVal buffer As String, ByVal buffsiz As Integer) As Integer
Declare Function DOCSFreeTrusteeHandle Lib "docsap32.dll" (ByVal htrustee As Integer) As Integer
Declare Function DOCSNewProjectHandle Lib "docsap32.dll" (ByVal libName As String, ByVal Project As String) As Integer
Declare Function DOCSNextProject Lib "docsap32.dll" (ByVal hProject As Integer, ByVal buffer As String, ByVal buffsiz As Integer) As Integer
Declare Function DOCSFreeProjectHandle Lib "docsap32.dll" (ByVal hProject As Integer) As Integer
Declare Function DOCSNewQuickSearchHandle Lib "docsap32.dll" (ByVal libName As String) As Integer
Declare Function DOCSNextQuickSearch Lib "docsap32.dll" (ByVal hqSearch As Integer, ByVal buffer As String, ByVal buffsiz As Integer) As Integer
Declare Function DOCSFreeQuickSearchHandle Lib "docsap32.dll" (ByVal hqSearch As Integer) As Integer

```

```

'Declare Function DOCSTrustee Lib "docsap32.dll" (ByVal trustee As String, ByVal docNum As String, ByVal libName As String) As Integer
Declare Function DOCSTrustee Lib "docsap32.dll" (ByVal trustee As String, ByVal docNum As String, ByVal libName As String, ByVal trusteeFlag As Integer) As Integer

```

```

Declare Function DOCSNewProfileKeywordHandle Lib "docsap32.dll" (ByVal docNum As String, ByVal libName As String) As Integer
Declare Function DOCSAddProfileKeyword Lib "docsap32.dll" (ByVal hkword As Integer, ByVal kword As String) As Integer
Declare Function DOCSSDeleteProfileKeyword Lib "docsap32.dll" (ByVal hkword As Integer, ByVal kword As String) As Integer
Declare Function DOCSNextProfileKeyword Lib "docsap32.dll" (ByVal hkword As Integer, ByVal buffer As String, ByVal buffsiz As Integer) As Integer
Declare Function DOCSFreeProfileKeywordHandle Lib "docsap32.dll" (ByVal hkword As Integer) As Integer
Declare Function DOCSNewDocument Lib "docsap32.dll" () As Integer

```

'PC DOCS API SQLView Object Prototypes

```

Declare Function DOCSSQLAddColumn Lib "docsap32.dll" (ByVal hSQLView As Integer, ByVal Column As String, ByVal ColType As Integer) As Integer
Declare Function DOCSSQLAddJoinColumn Lib "docsap32.dll" (ByVal hSQLView As Integer, ByVal Column As String, ByVal ColType As Integer, ByVal Table As String, ByVal flag As Integer) As Integer
Declare Function DOCSSQLDelete Lib "docsap32.dll" (ByVal hSQLView As Integer, ByVal whereClause As String) As Integer
Declare Function DOCSSQLFreeResultSet Lib "docsap32.dll" (ByVal hSQLView As Integer) As Integer
Declare Function DOCSSQLFreeView Lib "docsap32.dll" (ByVal hSQLView As Integer) As Integer
Declare Function DOCSSQLGetColumnValue Lib "docsap32.dll" (ByVal hSQLView As Integer, ByVal Column As String, ByVal buffer As String, ByVal bufsize As Integer) As Integer
Declare Function DOCSSQLGetError Lib "docsap32.dll" (ByVal hSQLView As Integer) As Integer
Declare Function DOCSSQLInsert Lib "docsap32.dll" (ByVal hSQLView As Integer) As Integer
Declare Function DOCSSQLNewView Lib "docsap32.dll" (ByVal Library As String, ByVal Table As String) As Integer
Declare Function DOCSSQLNextRow Lib "docsap32.dll" (ByVal hSQLView As Integer) As Integer
Declare Function DOCSSQLSelect Lib "docsap32.dll" (ByVal hSQLView As Integer) As Integer
Declare Function DOCSSQLSetColumnValue Lib "docsap32.dll" (ByVal hSQLView As Integer, ByVal Column As String, ByVal Value As String) As Integer
Declare Function DOCSSQLUpdate Lib "docsap32.dll" (ByVal hSQLView As Integer, ByVal whereClause As String) As Integer

```

' Document Profile Interface Prototypes

```

Declare Function DOCSAddProfileComponent Lib "docsap32.dll" (ByVal hProfile As Integer, ByVal componentPath As String) As Integer
Declare Function DOCSFreeProfile Lib "docsap32.dll" (ByVal hProfile As Integer) As Integer
Declare Function DOCSGetProfileError Lib "docsap32.dll" (ByVal hProfile As Integer) As Integer
Declare Function DOCSNewProfile Lib "docsap32.dll" (ByVal libName As String) As Integer
Declare Function DOCSaveProfile Lib "docsap32.dll" (ByVal hProfile As Integer, ByVal flags As Integer, ByVal docPath As String) As Integer
Declare Function DOCSSetProfileValue Lib "docsap32.dll" (ByVal hProfile As Integer, ByVal FieldName As String, ByVal Value As String) As Integer

```

' Utility Interface Prototypes

```

Declare Function DOCSAddPrintActivity Lib "docsap32.dll" (ByVal APPID As String, ByVal FieldName As String, ByVal pages As Integer, ByVal billable As String) As Integer
Declare Function DOCSAppExit Lib "docsap32.dll" (ByVal APPID As String) As Integer
Declare Function DOCSAssignFileACL Lib "docsap32.dll" (ByVal APPID As String, ByVal fileName As String) As Integer

```



```

ring) As Integer
Declare Function DOCSGetCurrentLibrary Lib "docsap32.dll" (ByVal libName As String) As Integer
'Declare Function DOCSGetNetUserID Lib "docsap32.dll" Alias "DOCSGetNetUserId" (ByVal userid As String) As Integer
Declare Function DOCSGetNetUserID Lib "docsap32.dll" (ByVal userid As String) As Integer

Declare Function DOCSGetMessageText Lib "docsap32.dll" (ByVal msgId As String, ByVal buffer As String, ByVal bufsize As Integer) As Integer
Declare Function DOCSGetProfileInfo Lib "docsap32.dll" (ByVal openFileID As String, ByVal item As String, ByVal buffer As String, ByVal bufsize As Integer) As Integer
Declare Function DOCSNewOrgHierarchyHandle Lib "docsap32.dll" (ByVal orgParent As String, ByVal LibraryName As String) As Integer
Declare Function DOCSNextOrgHierarchyChild Lib "docsap32.dll" (ByVal hSQLView As Integer, ByVal buffer As String, ByVal bufsize As Integer) As Integer
Declare Function DOCSFreeOrgHierarchyHandle Lib "docsap32.dll" (ByVal hOrgHierarchy As Integer) As Integer
Declare Function DOCSDisplayHistory Lib "docsap32.dll" (ByVal fileName As String) As Integer

Declare Function DOCSPrintDocument Lib "docsap32.dll" (ByVal docNum As String, ByVal version As String, ByVal Library As String) As Integer
Declare Function DOCSSetProfileDefault Lib "docsap32.dll" (ByVal item As String, ByVal Value As String, ByVal disable As Integer) As Integer
Declare Function DOCSShowProfile Lib "docsap32.dll" (ByVal APPID As String, ByVal openFileID As String, ByVal AllowEdits As Integer) As Integer
Declare Function DOCSSwitchToDesktop Lib "docsap32.dll" () As Integer
Declare Function DOCSViewDocument Lib "docsap32.dll" (ByVal docNum As String, ByVal version As String, ByVal Library As String) As Integer
Declare Function PerformLookup Lib "docsap32.dll" Alias "PerformLookUp" (ByVal libName As String, ByVal lookupId As String, ByVal sqlItem As String, ByVal sqlFilter As String, ByVal buffer As String) As Integer
Declare Function RemoveCloseOption Lib "docsap32.dll" (ByVal menuNumber As Integer) As Integer

'set error added by Detrick
Declare Function setErrorMessages Lib "docsap32.dll" (ByVal msg As Boolean) As Boolean
Declare Function IsDOCSRunning Lib "docsap32.dll" () As Integer 'REM FOR TESTING SP4

'MS Windows API Function Prototypes
Declare Function FindWindow Lib "user32" Alias "FindWindowA" (ByVal lpClassName As String, ByVal lpWindowName As String) As Long
Declare Function GetPrivateProfileString Lib "Kernel32" (ByVal lpApplicationName As String, lpKeyName As Any, ByVal lpDefault As String, ByVal lpReturnedString As String, ByVal nSize As Integer, ByVal lpFileName As String) As Integer
Declare Function WritePrivateProfileString Lib "Kernel32" (ByVal lpApplicationName As String, lpKeyName As Any, lpString As Any, ByVal lpFileName As String) As Integer

#ElseIf Win16 Then

'PC DOCS API Function Prototypes
'DMS Interface
Declare Function DOCSAddDocumentComponent Lib "docsapi.dll" (ByVal APPID As String, ByVal FileIn As String, ByVal componentPath As String) As Integer
Declare Function DOCSCloseDocument Lib "docsapi.dll" (ByVal APPID As String, ByVal fileName As String) As Integer
Declare Function DOCSOpenDocument Lib "docsapi.dll" (ByVal APPID As String, ByVal fileName As String) As Integer
Declare Function DOCSOpenDocumentReadOnly Lib "docsapi.dll" (ByVal APPID As String, ByVal fileName As String) As Integer
Declare Function DOCSOpenSelectedDocument Lib "docsapi.dll" (ByVal docNum As String, ByVal libName As String, ByVal APPID As String, ByVal docPath As String) As Integer
Declare Function DOCSOpenSelectedDoc Lib "docsapi.dll" (ByVal docNum As String, ByVal libName As String, ByVal appName As String, ByVal flags As Integer, ByVal fileName As String) As Integer
Declare Function DOCSQBEOpenDocument Lib "docsapi.dll" (ByVal criteria As String, ByVal APPID As String, ByVal openFileID As String) As Integer
Declare Function DOCSQBESelectDocument Lib "docsapi.dll" (ByVal criteria As String, ByVal docNum As String, ByVal libName As String) As Integer
Declare Function DOCSReleaseReadOnlyDocument Lib "docsapi.dll" (ByVal APPID As String, ByVal fileName As String) As Integer
Declare Function DOCSSaveAsAttachment Lib "docsapi.dll" (ByVal APPID As String, ByVal openFileID As String, ByVal flags As Integer, ByVal versionLabel As String, ByVal docPath As String, ByVal showPath As String) As Integer

```

```

Declare Function DOCSSaveDocAsExt Lib "docsapi.dll" (ByVal APPID As String, ByVal fileName As String,
ByVal flags As Integer, ByVal docPath As String, ByVal shadow As String) As Integer
Declare Function DOCSSaveDocument Lib "docsapi.dll" (ByVal APPID As String, ByVal fileName As String,
ByVal docPath As String, ByVal shadow As String) As Integer
Declare Function DOCSSaveDocumentAs Lib "docsapi.dll" (ByVal APPID As String, ByVal fileName As String,
ByVal docPath As String, ByVal shadow As String) As Integer
Declare Function DOCSSelectDocument Lib "docsapi.dll" (ByVal docNum As String, ByVal libName As String)
As Integer
Declare Function DOCSSaveDoc Lib "docsapi.dll" (ByVal APPID As String, ByVal fileName As String,
ByVal flag As Integer, ByVal docPath As String, ByVal shadow As String) As Integer
Declare Function DOCSSaveDocAs Lib "docsapi.dll" (ByVal APPID As String, ByVal fileName As String,
ByVal flag As Integer, ByVal docPath As String, ByVal shadow As String) As Integer

```

'NEW or updated FUNCTIONS

```

Declare Function DOCSOpenSelectedDoc Lib "docsapi.dll" (ByVal docNum As String, ByVal libName As String,
ByVal APPID As String, ByVal flags As Integer, ByVal docPath As String) As Integer
Declare Function DOCSOpenSelectedVersion Lib "docsapi.dll" (ByVal docNum As String, ByVal libName
As String, ByVal verslabel As String, ByVal APPID As String, ByVal flags As Integer, ByVal docPath
As String) As Integer
Declare Function DOCSAssignFileACLFromDB Lib "docsapi.dll" (ByVal APPID As String, ByVal fileName
As String, ByVal readDB As Integer) As Integer
Declare Function DOCSQBESearch Lib "docsapi.dll" (ByVal docNum As String, ByVal docPath As String)
As Integer
Declare Function DOCSSaveAsNewSubVersion Lib "docsapi.dll" (ByVal APPID As String, ByVal fileName
As String, ByVal comments As String, ByVal Author As String, ByVal Typist As String, ByVal billable
As Byte, ByVal ShowForm As Integer, ByVal docPath As String, ByVal shadow As String) As Integer
Declare Function DOCSSaveAsNewVersion Lib "docsapi.dll" (ByVal APPID As String, ByVal fileName As
String, ByVal comments As String, ByVal Author As String, ByVal Typist As String, ByVal billable
As Byte, ByVal ShowForm As Integer, ByVal docPath As String, ByVal shadow As String) As Integer
Declare Function DOCSDeleteDocument Lib "docsapi.dll" (ByVal docNum As String, ByVal libName As String,
ByVal flags As Integer) As Integer
Declare Function DOCSAddRouteActivity Lib "docsapi.dll" (ByVal docNum As Long, ByVal libName As String)
As Integer
Declare Function DOCSSelectProfileForm Lib "docsapi.dll" (ByVal libName As String, ByVal APPID As
String, ByVal flags As Integer) As Integer
Declare Function DOCSSelectQBForm Lib "docsapi.dll" (ByVal flags As Integer) As Integer
Declare Function DOCSEnableErrorMessage Lib "docsapi.dll" (ByVal flags As Integer) As Integer
Declare Function DOCSAddTrustee Lib "docsapi.dll" (ByVal trustee As String, ByVal docNum As String,
ByVal libName As String, ByVal rights As Integer, ByVal flags As Integer) As Integer
Declare Function DOCSRemoveTrustee Lib "docsapi.dll" (ByVal trustee As String, ByVal docNum As String,
ByVal libName As String, ByVal flags As Integer) As Integer
Declare Function DOCSNewTrusteeHandle Lib "docsapi.dll" (ByVal docNum As String, ByVal libName As
String) As Integer
Declare Function DOCSNextTrustee Lib "docsapi.dll" (ByVal htrustee As Integer, ByVal buffer As String,
ByVal buffsiz As Integer) As Integer
Declare Function DOCSFreeTrusteeHandle Lib "docsapi.dll" (ByVal htrustee As Integer) As Integer
Declare Function DOCSNewProjectHandle Lib "docsapi.dll" (ByVal libName As String, ByVal Project As
String) As Integer
Declare Function DOCSNextProject Lib "docsapi.dll" (ByVal hProject As Integer, ByVal buffer As String,
ByVal buffsiz As Integer) As Integer
Declare Function DOCSFreeProjectHandle Lib "docsapi.dll" (ByVal hProject As Integer) As Integer
Declare Function DOCSNewQuickSearchHandle Lib "docsapi.dll" (ByVal libName As String) As Integer
Declare Function DOCSNextQuickSearch Lib "docsapi.dll" (ByVal hqSearch As Integer, ByVal buffer As
String, ByVal buffsiz As Integer) As Integer
Declare Function DOCSFreeQuickSearchHandle Lib "docsapi.dll" (ByVal hqSearch As Integer) As Integer
Declare Function DOCSTrustee Lib "docsapi.dll" (ByVal trustee As String, ByVal docNum As String,
ByVal libName As String) As Integer
Declare Function DOCSNewProfileKeywordHandle Lib "docsapi.dll" (ByVal docNum As String, ByVal libName
As String) As Integer
Declare Function DOCSAddProfileKeyword Lib "docsapi.dll" (ByVal hkword As Integer, ByVal kword As
String) As Integer
Declare Function DOCSDeleteProfileKeyword Lib "docsapi.dll" (ByVal hkword As Integer, ByVal kword
As String) As Integer
Declare Function DOCSNextProfileKeyword Lib "docsapi.dll" (ByVal hkword As Integer, ByVal buffer As
String, ByVal buffsiz As Integer) As Integer
Declare Function DOCSFreeProfileKeywordHandle Lib "docsapi.dll" (ByVal hkword As Integer) As Integer
Declare Function DOCSNewDocument Lib "docsapi.dll" () As Integer

```

'PC DOCS API SQLView Object Prototypes

```

Declare Function DOCSSQLAddColumn Lib "docsapi.dll" (ByVal hSQLView As Integer, ByVal Column As String, ByVal ColType As Integer) As Integer
Declare Function DOCSSQLAddJoinColumn Lib "docsapi.dll" (ByVal hSQLView As Integer, ByVal Column As String, ByVal ColType As Integer, ByVal Table As String, ByVal flag As Integer) As Integer
Declare Function DOCSSQLDelete Lib "docsapi.dll" (ByVal hSQLView As Integer, ByVal whereClause As String) As Integer
Declare Function DOCSSQLFreeResultSet Lib "docsapi.dll" (ByVal hSQLView As Integer) As Integer
Declare Function DOCSSQLFreeView Lib "docsapi.dll" (ByVal hSQLView As Integer) As Integer
Declare Function DOCSSQLGetColumnValue Lib "docsapi.dll" (ByVal hSQLView As Integer, ByVal Column As String, ByVal buffer As String, ByVal bufsize As Integer) As Integer
Declare Function DOCSSQLGetError Lib "docsapi.dll" (ByVal hSQLView As Integer) As Integer
Declare Function DOCSSQLInsert Lib "docsapi.dll" (ByVal hSQLView As Integer) As Integer
Declare Function DOCSSQLNewView Lib "docsapi.dll" (ByVal Library As String, ByVal Table As String) As Integer
Declare Function DOCSSQLNextRow Lib "docsapi.dll" (ByVal hSQLView As Integer) As Integer
Declare Function DOCSSQLSelect Lib "docsapi.dll" (ByVal hSQLView As Integer) As Integer
Declare Function DOCSSQLSetColumnValue Lib "docsapi.dll" (ByVal hSQLView As Integer, ByVal Column As String, ByVal Value As String) As Integer
Declare Function DOCSSQLUpdate Lib "docsapi.dll" (ByVal hSQLView As Integer, ByVal whereClause As String) As Integer

```

' Document Profile Interface Prototypes

```

Declare Function DOCSAddProfileComponent Lib "docsapi.dll" (ByVal hProfile As Integer, ByVal componentPath As String) As Integer
Declare Function DOCSFreeProfile Lib "docsapi.dll" (ByVal hProfile As Integer) As Integer
Declare Function DOCSGetProfileError Lib "docsapi.dll" (ByVal hProfile As Integer) As Integer
Declare Function DOCSNewProfile Lib "docsapi.dll" (ByVal libName As String) As Integer
Declare Function DOCSSaveProfile Lib "docsapi.dll" (ByVal hProfile As Integer, ByVal flags As Integer, ByVal docPath As String) As Integer
Declare Function DOCSSetProfileValue Lib "docsapi.dll" (ByVal hProfile As Integer, ByVal fieldName As String, ByVal Value As String) As Integer

```

' Utility Interface Prototypes

```

Declare Function DOCSAddPrintActivity Lib "docsapi.dll" (ByVal APPID As String, ByVal fieldName As String, ByVal pages As Integer, ByVal billable As String) As Integer
Declare Function DOCSAppExit Lib "docsapi.dll" (ByVal APPID As String) As Integer
Declare Function DOCSAssignFileACL Lib "docsapi.dll" (ByVal APPID As String, ByVal fileName As String) As Integer
Declare Function DOCSDisplayHistory Lib "docsapi.dll" (ByVal fileName As String) As Integer
Declare Function DOCSGetCurrentLibrary Lib "docsapi.dll" (ByVal libName As String) As Integer
Declare Function DOCSGetNetUserID Lib "docsapi.dll" (ByVal userid As String) As Integer
Declare Function DOCSGetMessageText Lib "docsapi.dll" (ByVal msgId As String, ByVal buffer As String, ByVal bufsize As Integer) As Integer
Declare Function DOCSGetProfileInfo Lib "docsapi.dll" (ByVal openFileID As String, ByVal item As String, ByVal buffer As String, ByVal bufsize As Integer) As Integer

```

```

Declare Function DOCSNewOrgHierarchyHandle Lib "docsapi.dll" (ByVal orgParent As String, ByVal LibraryName As String) As Integer
Declare Function DOCSNextOrgHierarchyChild Lib "docsapi.dll" (ByVal hSQLView As Integer, ByVal buffer As String, ByVal bufsize As Integer) As Integer
Declare Function DOCSFreeOrgHierarchyHandle Lib "docsapi.dll" (ByVal hOrgHierarchy As Integer) As Integer

```

```

Declare Function DOCSPrintDocument Lib "docsapi.dll" (ByVal docNum As String, ByVal version As String, ByVal Library As String) As Integer
Declare Function DOCSSetProfileDefault Lib "docsapi.dll" (ByVal item As String, ByVal Value As String, ByVal disable As Integer) As Integer
Declare Function DOCSShowProfile Lib "docsapi.dll" (ByVal APPID As String, ByVal openFileID As String, ByVal AllowEdits As Integer) As Integer
Declare Function DOCSSwitchToDesktop Lib "docsapi.dll" () As Integer
Declare Function DOCSViewDocument Lib "docsapi.dll" (ByVal docNum As String, ByVal version As String, ByVal Library As String) As Integer
Declare Function PerformLookup Lib "docsapi.dll" (ByVal libName As String, ByVal lookupId As String, ByVal sqlItem As String, ByVal sqlFilter As String, ByVal buffer As String) As Integer
Declare Function RemoveCloseOption Lib "docsapi.dll" (ByVal menuNumber As Integer) As Integer
Declare Function DOCSRemoveSecurity Lib "docsapi.dll" (ByVal docNum As String, ByVal libName As String) As Integer

```

set error added by Detrick

```
Declare Function setErrorMessages Lib "docsapi.dll" (ByVal msg As Boolean) As Boolean
Declare Function IsDOCSRunning Lib "docsapi.dll" () As Integer 'REM FOR TESTING SP4
```

```
'MS Windows API Function Prototypes
```

```
Declare Function FindWindow Lib "user" (ByVal lpClassName As Any, ByVal lpWindowName As Any) As Integer
```

```
Declare Function GetPrivateProfileString Lib "Kernel" (ByVal lpApplicationName As String, lpKeyName As Any, ByVal lpDefault As String, ByVal lpReturnedString As String, ByVal nSize As Integer, ByVal lpFileName As String) As Integer
```

```
Declare Function WritePrivateProfileString Lib "Kernel" (ByVal lpApplicationName As String, lpKeyName As Any, lpString As Any, ByVal lpFileName As String) As Integer
```

```
#End If
```

```
'Lookup test function
```

```
'Declare Function LTEST1 Lib "LookTest" (ByVal buffer As String, ByVal bufsize As Integer, ByVal qbeMode As Integer) As Integer
```

'PC DOCS API Function Prototypes

```
'Declare Function DOCSAppExit Lib "DOCSAPI.DLL" (ByVal AppId As String) As Integer
'Declare Function DOCSAssignFileACL Lib "DOCSAPI.DLL" (ByVal AppId As String, ByVal Filename As String) As Integer
'Declare Function DOCSOpenDocument Lib "DOCSAPI.DLL" (ByVal AppId As String, ByVal Filename As String) As Integer
'Declare Function DOCSCloseDocument Lib "DOCSAPI.DLL" (ByVal AppId As String, ByVal Filename As String) As Integer
'Declare Function DOCSSaveDocument Lib "DOCSAPI.DLL" (ByVal AppId As String, ByVal Filename As String, ByVal DocPath As String, ByVal shadow As String) As Integer
'Declare Function docssavedocas Lib "DOCSAPI.DLL" (ByVal AppId As String, ByVal Filename As String, ByVal flags As Integer, ByVal DocPath As String, ByVal shadow As String) As Integer
'Declare Function DOCSReleaseReadOnlyDocument Lib "DOCSAPI.DLL" (ByVal AppId As String, ByVal Filename As String) As Integer
'Declare Function DOCSNewProfile Lib "DOCSAPI.DLL" (ByVal libName As String) As Integer
'Declare Function DOCSSetProfileValue Lib "DOCSAPI.DLL" (ByVal hProfile As Integer, ByVal FieldName As String, ByVal Value As String) As Integer
'Declare Function DOCSSaveProfile Lib "DOCSAPI.DLL" (ByVal hProfile As Integer, ByVal flags As Integer, ByVal DocPath As String) As Integer
'Declare Function DOCSFreeProfile Lib "DOCSAPI.DLL" (ByVal hProfile As Integer) As Integer
'Declare Function DOCSSaveAsAttachment Lib "DOCSAPI.DLL" (ByVal AppId As String, ByVal openFileID As String, ByVal flags As Integer, ByVal versionLabel As String, ByVal DocPath As String, ByVal shadowPath As String) As Integer
'Declare Function DOCSQBESelectDocument Lib "DOCSAPI.DLL" (ByVal criteria As String, ByVal docNum As String, ByVal libName As String) As Integer
'Declare Function DOCSQBEOpenDocument Lib "DOCSAPI.DLL" (ByVal criteria As String, ByVal AppId As String, ByVal openFileID As String) As Integer
'Declare Function DOCSSelectDocument Lib "DOCSAPI.DLL" (ByVal docNum As String, ByVal libName As String) As Integer
'Declare Function DOCSSaveDoc Lib "DOCSAPI.DLL" (ByVal AppId As String, ByVal Filename As String, flag As Integer, ByVal DocPath As String, ByVal shadow As String) As Integer
'Declare Function DOCSAddDocumentComponent Lib "DOCSAPI.DLL" (ByVal AppId As String, ByVal Filename As String, ByVal componentPath As String) As Integer
'Declare Function DOCSGetCurrentLibrary Lib "DOCSAPI.DLL" (ByVal libName As String) As Integer
'Declare Function DOCSGetNetUserID Lib "DOCSAPI.DLL" (ByVal UserID As String) As Integer
'Declare Function DOCSPrintDocument Lib "DOCSAPI.DLL" (ByVal docNum As String, ByVal version As String, ByVal Library As String) As Integer
'Declare Function DOCSViewDocument Lib "DOCSAPI.DLL" (ByVal docNum As String, ByVal version As String, ByVal Library As String) As Integer
'Declare Function PerformLookUp Lib "DOCSAPI.DLL" (ByVal libName As String, ByVal lookupId As String, ByVal sqlItem As String, ByVal sqlFilter As String, ByVal buffer As String) As Integer
```

'PC DOCS API SQLView Object Prototypes

```
'Declare Function DOCSSQLNewView Lib "DOCSAPI.DLL" (ByVal Library As String, ByVal Table As String) As Integer
'Declare Function DOCSSQLAddColumn Lib "DOCSAPI.DLL" (ByVal hSqlView As Integer, ByVal Column As String, ByVal ColType As Integer) As Integer
'Declare Function DOCSSQLAddJoinColumn Lib "DOCSAPI.DLL" (ByVal hSqlView As Integer, ByVal Column As String, ByVal ColType As Integer, ByVal Table As String, ByVal flag As Integer) As Integer
'Declare Function DOCSSQLSetColumnValue Lib "DOCSAPI.DLL" (ByVal hSqlView As Integer, ByVal Column As String, ByVal Value As String) As Integer
'Declare Function DOCSSQLSelect Lib "DOCSAPI.DLL" (ByVal hSqlView As Integer) As Integer
'Declare Function DOCSSQLUpdate Lib "DOCSAPI.DLL" (ByVal hSqlView As Integer, ByVal WhereClause As String) As Integer
'Declare Function DOCSSQLDelete Lib "DOCSAPI.DLL" (ByVal hSqlView As Integer, ByVal WhereClause As String) As Integer
'Declare Function DOCSSQLInsert Lib "DOCSAPI.DLL" (ByVal hSqlView As Integer) As Integer
'Declare Function DOCSSQLGetError Lib "DOCSAPI.DLL" (ByVal hSqlView As Integer) As Integer
'Declare Function DOCSSQLGetColumnValue Lib "DOCSAPI.DLL" (ByVal hSqlView As Integer, ByVal Column As String, ByVal buffer As String, ByVal BufSize As Integer) As Integer
'Declare Function DOCSSQLNextRow Lib "DOCSAPI.DLL" (ByVal hSqlView As Integer) As Integer
'Declare Function DOCSSQLFreeResultSet Lib "DOCSAPI.DLL" (ByVal hSqlView As Integer) As Integer
'Declare Function DOCSSQLFreeView Lib "DOCSAPI.DLL" (ByVal hSqlView As Integer) As Integer
```

'MS Windows API Function Prototypes

```
'Declare Function FindWindow Lib "User" (ByVal lpClassName As Any, ByVal lpWindowName As Any) As Integer
```

'Some Constants that I might use

```
Global Const MB_OK = 0, MB_OKCANCEL = 1
```

Module1 - 2

```
Global Const MB_YESNOCANCEL = 3, MB_YESNO = 4
Global Const MB_ICONSTOP = 16, MB_ICONQUESTION = 32
Global Const MB_ICONEXCLAM = 48, MB_ICONINFORMATION = 64
Global Const IDNO = 7
Global Const API_STOP = MB_OK + MB_ICONSTOP
Global Const API_WHAT = MB_OK + MB_ICONQUESTION
Global Const API_GREAT = MB_OK + MB_ICONEXCLAM
```

```
Global Const COLTYPE_DATE = 0
Global Const COLTYPE_TIME = 1
Global Const COLTYPE_STRING = 2
Global Const COLTYPE_INT = 3
Global Const COLTYPE_CHAR = 4
Global Const COLTYPE_FOREIGNKEY = 5
```



IntelliGard™

Auto Encryption Manager

Selection Criteria	Value	Key ID
List1		

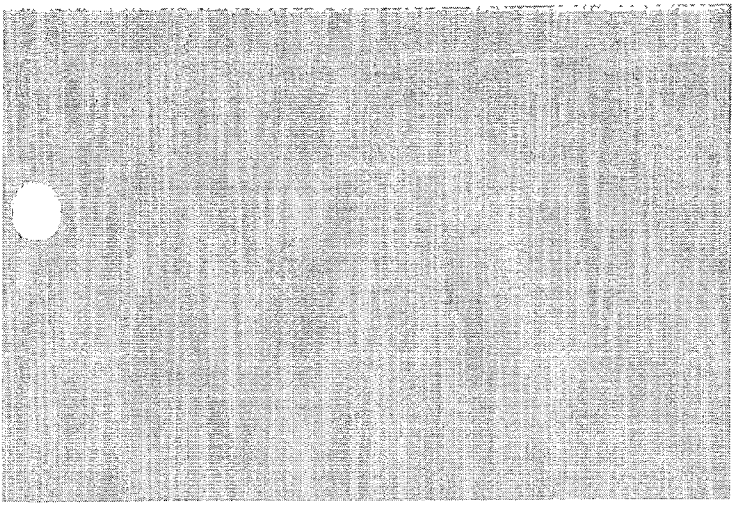
Add

Modify

Delete

Close

Intelligard
to
DOCS,
Prototypes



(0)

(0)





Group Encryption

Encryption Key Selection

☐

Combo1

☐

Encrypt

Close

Group Selection Criteria

Group Decrypt



Selection Criteria

Decrypt

Close


EXHIBIT Z-24

PC DOCS EDM

Programmers Reference

(1996)

Declaration of Stephen Zizzi



Document Management for the Enterprise . . . and Beyond!

PC DOCS

DOCS Designer™ and Data Dictionary

. . . .

DOCUMENT
ORGANIZATION &
CONTROL
SYSTEM

DOCS OPEN™

. . . .

FROM PC DOCS, INC.

Published by
PC DOCS Inc.
A PC DOCS Group International Company
25 Burlington Mall Rd., Burlington, MA 01803
(617) 273-3800 Sales and Administration
(617) 272-3693 Fax

Copyright

© PC DOCS Inc 1996. All Rights Reserved. No part of the contents of this book or the accompanying CD may be reproduced or transmitted in any form or by any means without prior written permission of PC DOCS Inc. The information contained in this document is subject to change without notice.

Your license agreement with PC DOCS INC, included with the product, specifies the permitted and prohibited uses of the product. Any unauthorized duplication or use of the DOCS Open product in whole or in part is strictly forbidden.

Third Edition: May 1996

Trademarks

DOCS, DOCS Open, DOCS Designer, DOCS Interchange, DOCS Unplugged, and DO-IT are trademarks of PC DOCS Inc.

All other trademarks referred to herein are trademarks of their respective companies.

Support and Training

PC DOCS Inc. is dedicated to customer service. If you ever need assistance using our program, call the technical support hotline in your area.

United States/Canada - PC DOCS, Inc. 8:00 a.m. to 8:00 p.m. E.S.T.

Phone (904) 942-5000

Fax (904) 942-8085

United Kingdom - Quintec International Limited

Phone 44 (0) 1268 270601

Fax 44 (0) 1268 270602

Pacific/Asia - Quorum Growth

Phone (65) 463 2633

Fax (65) 463-2295

Australia/New Zealand - Educom

Phone 02 957-5833

Fax 02 957-3128

PC DOCS, Inc. provides high quality user, administrator, and developer training. Classes are offered at authorized training centers world-wide. For contact and scheduling information, please contact the PC DOCS Training Coordinator at 617-273-3800.

The Menus

The DOCS Designer menus allow you to perform a variety of functions.

The File Menu

Clicking once on the File menu displays the following menu items:

<u>F</u> ile	<u>E</u> dit	<u>O</u> ptions	<u>L</u> ibrary
Open <u>P</u> rofile Form...			
Open Search Result <u>F</u> orm...			
Open <u>M</u> aintenance Form...			
Close			
Save			
Save as			
<u>O</u> pen From Disk...			
Save to Disk...			
Save <u>L</u> ookups to Disk...			
<u>R</u> eplicate Library Design...			
Exit			

- ☐ **Open Profile Form...:** Selecting the Open Profile Form menu item displays a list of all Profile forms. You may edit or copy any of the displayed forms. For more information on altering forms, refer to Chapter 2 of this manual.
- ☐ **Open Search Results Form...:** Selecting the Open Search Results Form menu item displays a list of all Search Results forms. You may edit or copy any of the displayed forms. For more information on altering forms, refer to Chapter 2 of this manual.
- ☐ **Open Maintenance Form...:** Selecting the Open Maintenance Form menu item displays a list of all editable Maintenance forms. You may edit or copy any of the displayed forms. For more information on altering forms, refer to Chapter 2 of this manual.
- ☐ **Close:** Closes the currently open form.
- ☐ **Save:** Save changes to the current form.
- ☐ **Save As:** Save changes to the current form as a new form.

- ☐ **Open from Disk...:** Selecting the Open from Disk menu item displays the standard File Open dialog. You may retrieve any form saved to a file. All default DOCS Open forms are available in the DOCS Open program directory.
- ☐ **Save to Disk...:** Selecting the Save to Disk menu item displays the standard File Save dialog. You may save any form to a disk. This form can be opened and used at another site. You should save the form with the .FRM extension.
- ☐ **Save Lookups to Disk:** Selecting the Save Lookups to Disk menu item displays a list of available lookups. You may select one or more of these lookups to save to disk.
- ☐ **Replicate Library Design:** Selecting the Replicate Library Design menu item displays a list of defined Remote Libraries. You may select the libraries to copy form and database changes to. Refer to Chapter 4 of this manual for more information.
- ☐ **Exit:** Selecting this option exits the DOCS Designer. You are returned to the DOCS Open desktop or the Windows desktop.

The Edit Menu

Clicking once on the Edit menu displays the following menu items;

<u>E</u> dit	<u>O</u> ptions	<u>L</u> ibrary
Cut		Del
<u>C</u> opy		Ctrl+Ins
<u>P</u> aste		Shift+Ins
Form Attributes...		
Duplicate <u>O</u> bject		
Edit Table <u>L</u> ookups...		
Edit <u>D</u> ata Dictionary...		

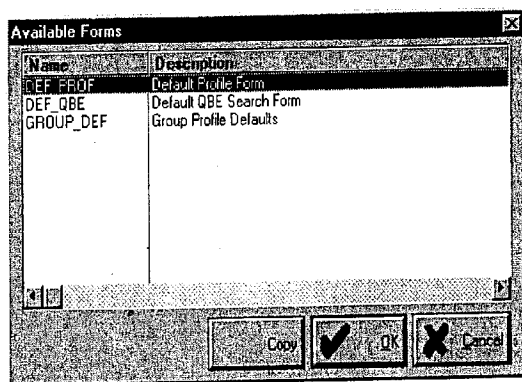
- ☐ **Cut:** Selecting the Cut menu item places the currently selected form object into the Windows Clipboard.
- ☐ **Copy:** Selecting the Copy menu item copies the currently selected form object into the Windows Clipboard.
- ☐ **Paste:** Selecting the Paste menu item copies the contents of the Windows Clipboard onto the selected area of the form.
- ☐ **Form Attributes:** Selecting the Form Attributes menu item displays the form attributes for the currently open form.

Opening a Form for Editing

Before you can make changes to a form in Designer, you must open the form. When you open a form in Designer you may open the original form or a copy of the form. PC DOCS, Inc. recommends that you never make changes to the original forms.

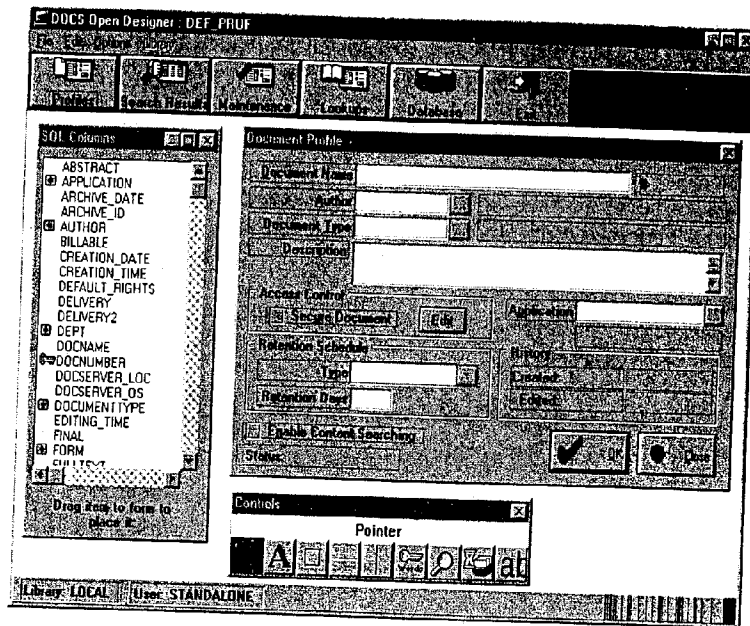
To open a form in DOCS Designer:

1. Click on the Open Profile Form, Open Search Result Form, or Open Maintenance Form menu item. Alternatively, you may click the Profile, Search Results, or Maintenance button on the DOCS Designer button bar.
2. The Available Forms window is displayed. Highlight the form you wish to edit and select Copy or OK.

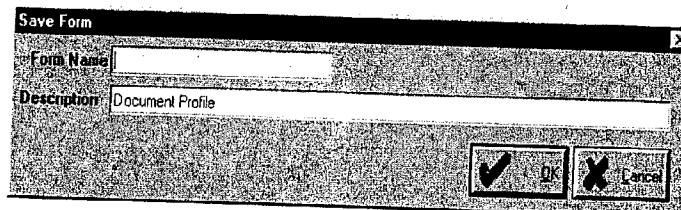


Warning! You should always Copy an original form. If you make a mistake in the form you are designing, you will be able to retrieve the original form. There is no Undo option in the DOCS Designer.

3. The selected form is displayed on the DOCS Designer desktop. The SQL Columns list and Controls window are also displayed.



4. If you did not choose to copy the original form in Step 2 of these instructions and you do not want to overwrite the original form, choose **Save As** from the **File** menu before making any changes to the form.
5. If you chose **Save As**, the following screen is displayed.



- ☐ **Form Name:** Enter a form name, up to 18 characters (do not use spaces).
- ☐ **Description:** Enter a form description, up to 70 characters (spaces are allowed). This description will be displayed in the title bar of the form. The default description is "Document Profile."

Click OK. You are returned to the DOCS Designer desktop.



You may edit the Narrow object just like any other push button object. Refer to the section earlier in this chapter for more information on push button attributes.

Save



The Save object appears on the form as a push button. A user may select this button to create a Quick Search using the criteria of the current document listing. The Save push button functions identically to the Save As Quick Search menu item on the Search menu. The following is an example of a Save push button.



You may edit the Save object just like any other push button object. Refer to the section earlier in this chapter for more information on push button attributes.

Hit



The Hit object appears as a read only edit field. The field displays the hit number for the current record. The field is read only and is for informational purposes only. The following is an example of the Hit field as it appears in Designer.



You may edit the Hit object just like any other edit field. Refer to the section earlier in this chapter for more information on edit field attributes.

Expand



The Expand object appears on the form as a push button. A user may select this button to add to the search criteria. The Expand push button functions identically to the Expand menu item on the Search menu. The following is an example of the Expand push button.

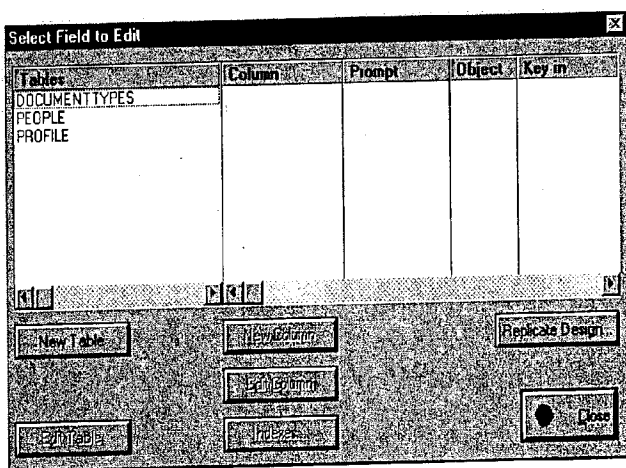


You may edit the Expand object just like any other push button object. Refer to the section earlier in this chapter for more information on push button attributes.

Accessing the Database

All database modifications are done through the Select Field to Edit window. There are two methods for accessing this window. The first method is to select the Edit Data Dictionary option from the Edit menu. The second method is to click the Database button on the Designer button bar.

Once you select the menu option or button the following window is displayed:



- ❑ **Table List:** The table list displays all user modifiable tables for the current library. The columns of the selected table are displayed in the Columns list.
- ❑ **Column List:** This list displays information about the columns of the selected table. If no table is selected in the Tables list, this area is blank. The following information is displayed in the list:
 - ✓ **Column:** The name of the column in the database.
 - ✓ **Prompt:** The prompt for the column when it is placed on a form.
 - ✓ **Object:** The object type of the column.
 - ✓ **Key in:** If the column is a foreign key, this indicates the table referenced by the foreign key.
- ❑ **New Table:** Use this button to define a new table for the database. For more information on creating new tables, refer to the section later in this chapter entitled "Add a New Table to the Database."

- ☐ **Edit Table:** Use this button to edit the description of an existing table. For more information on editing tables, refer to the section later in this chapter entitled "Edit Existing Tables and Columns."
- ☐ **New Column:** Use this button to define a new column in the selected table. For more information on creating new columns, refer to the section later in this chapter entitled "Add a New Column to a Table."
- ☐ **Edit Column:** Use this button to modify the definition for the selected column. For more information on editing columns, refer to the section later in this chapter entitled "Edit Existing Tables and Columns."
- ☐ **Indexes:** Use this button to create indexes for the selected table. For more information on indexes, refer to the section later in this chapter entitled "Indexes."
- ☐ **Replicate Design:** Use this button to replicate changes in this library to additional remote libraries. For more information on this feature, refer to the last section of this chapter entitled "Replicate Library Design."

Add a New Column to a Table

You may add new columns to existing tables or new tables. You must create at least one column before you can save a new table. To add a new column, select the table to which you want the column added from the Select Field to Edit window. Next, click the New Column button. The Edit Column Description window is displayed.

Complete the column description information. The description fields vary depending on the Object Type and Key values. The different description fields are shown below:

- ☐ **Column Name:** The name of the column in the SQL database. Alpha-numeric characters are allowed. Spaces are not allowed. Use the underscore character (_) to separate words instead of spaces.
- ☐ **Type:** Refers to the SQL data type of the column. Syntax for each data type will vary based on your SQL Server type. The following are available data types:
 - ✓ **String:** The default data type when creating a new column. This type is used for any combination of alpha-numeric characters.
 - ✓ **Date:** Allows only date entries in the column.
 - ✓ **Time:** Allows only time entries in the column.

- ✓ **Real:** Allows real numbers in the column.
- ✓ **Integer:** Allows integers in the column (whole numbers).

Note Date columns do not allow dates before 1900.

- ☐ **Key:** Designates a column as a candidate, foreign, or foreign/candidate key.
 - ✓ **Candidate Key:** A column that uniquely identifies the rows in a table. For example, in the PROFILE table, the DOCNUMBER column is a candidate key. Each document number is unique and will not be repeated; therefore it uniquely identifies each row in the PROFILE table.
 - ✓ **Primary Key:** A column that uniquely identifies the rows in a table. The primary key *must* be unique; for this reason, the primary key usually has a data type of Integer. Most primary keys in DOCS Open are identified by the column name SYSTEM_ID.
 - ✓ **Foreign Key:** A foreign key is a column in one table that refers a column in another table. The foreign key often has the name of the foreign table to which it relates. For example, the column DOCUMENTTYPE in the PROFILE table is a foreign key to the SYSTEM_ID column in the DOCUMENTTYPES table. The foreign key almost always relates to the primary key in another table; occasionally, a foreign key may relate to a candidate key instead.
 - ✓ **Foreign/Candidate Key:** This combines the attributes of a foreign and a candidate key. This key is required for any foreign keys in tables created by you that you wish to require on the Profile form and therefore be used during Document Import.
- ☐ **Length:** This field is only displayed if you selected a column type of string. This defines the length of the column in the table.
- ☐ **Object Type:** This field defines the way the column will appear and how it will operate on the form. The available object types are: Edit, Combo Box, Check Box, MultiEdit, Radio Group, and Buffer. The Edit, Combo Box, Check Box, MultiEdit and Radio Group object types are described in Chapter 2 of this manual. The Buffer object type is used for foreign and foreign/candidate keys only.
- ☐ **Help ID:** The name of the context sensitive help for the column.
- ☐ **Prompt:** The prompt for the column. The & character indicates which letter will be underlined for use with the keyboard when the form is in use on the DOCS Open Desktop.

Add a New Column to a Table

- ☐ **Object Items:** The list of items in the combo box list. Each item must be separated by a pipe (|) or be on a separate line. To move the cursor down a line, press CONTROL+ENTER. You may enter the values to actually store in the database or you may enter a separate value to display to the user. The database value and display value may be on the same line, separated by a semicolon (;). For example, the combo box object items for storage type are:

A;Archive|D;Delete|K;Keep|O;Optical|T;Template|P;Paper

In the above example, if a user selected the first option, "A" would be the text stored in the database. The word "Archive" would actually be displayed in the combo box list.

If you selected an Object Type of Check Box, the following field descriptions are displayed:

- ☐ **On Value:** The text stored in the database when the item is checked. This is usually "Y" or "1."
- ☐ **Off Value:** The text stored in the database when the item is not checked. This is usually "N" or "0."

If you selected an Object Type of Radio Group, the following field description is displayed:

- ☐ **Object Items:** The list of items in the combo box list. Each item must be separated by a pipe (|) or be on a separate line. To move the cursor down a line, press CONTROL+ENTER. You may enter the database value and display value on the same line separated by a semicolon (;). For example, the radio group object items for storage type are:

A;Archive|D;Delete|K;Keep|O;Optical|T;Template|P;Paper

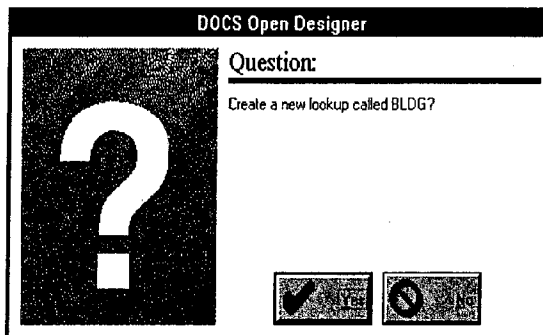
In the above example, if a user selected the first option, "A" would be the text stored in the database. The word "Archive" would actually be displayed next to the radio group option.

Saving the New Column

Once you have completed the definition of the column, click on the OK button at the bottom of the screen. This automatically adds the column to the selected table, and it will appear in the column list. To add the new column to form, refer to chapter 2 of this manual.

Save the New Table

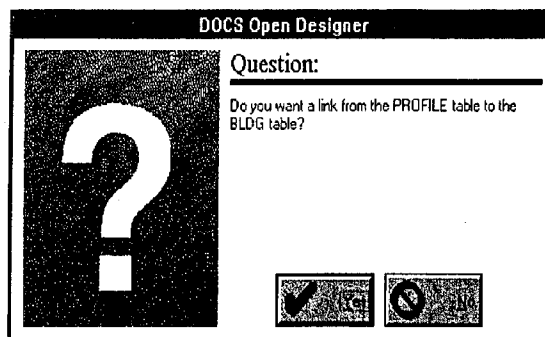
To save the table and its columns, click on the Save Table button. The table is automatically be saved to the database. You will then be given a screen asking you if you wish to create a Lookup for the new table, shown below:



You should click on the OK button to create the new lookup. Refer to the section on Lookups later in this manual for instructions on creating and editing lookups.

Link the New Table to the Profile Table

After saving the Lookup form, you will be shown the following screen asking if you wish to link the new table to the PROFILE table, shown below:



What is a Link?

By answering YES, you establish a link between the PROFILE table and the new table you have created (in the example given in this chapter, the BLDG table is the new table).

The link places a column in the PROFILE table with the name of the new table as the column name. This column in the PROFILE table is the foreign key to the new table, and this column points to the value stored as the primary key (the SYSTEM_ID) in the new table.

Using the example given in this chapter, the link created would be the new column BLDG in the PROFILE table, the foreign key to the BLDG table that relates to the SYSTEM_ID column in the BLDG table (PROFILE.BLDG=BLDG.SYSTEM_ID).

When to Link?

- ☐ **Direct Relationships to PROFILE:** Some tables are linked directly to the Profile table. For example, you might create an ACCOUNT table in order to associate a Profile with a specific Account. In this case, you would answer YES and link the table to the Profile table when so asked.
- ☐ **Non-Direct Relationships:** Another type of table is one that relates to *another* table that is linked to the PROFILE table. For example, the BLDG table contains information about Buildings where PEOPLE reside. Therefore, instead of linking BLDG to the PROFILE table, you would add a column to the PEOPLE table that serves as a foreign key to the BLDG table. PEOPLE is linked to PROFILE (for Author, Typist, Last Edited By, etc.), but BLDG is not. So you would answer NO when linking a table that serves the same function as BLDG.
- ☐ **Hierarchical Relationships:** When creating tables with a hierarchical nature, instead of linking each new table that you add to the database, you will only link the *last* table in the hierarchical chain. Using the example of tables named GRANDPARENT, PARENT, and CHILD, let's explore why.

When creating a Profile form, you would select columns from tables to appear on the form: the TYPE_ID from the DOCUMENTTYPES table, and the USER_ID from the PEOPLE for Author. To include information from the hierarchical tables, you might think that each table would need a foreign key in the PROFILE table; but this is not necessary and is in fact redundant.

The third table in the chain (in our example, CHILD) will contain references to both tables ranked above it (in our example, PARENT and GRANDPARENT). The CHILD table will have columns that are foreign keys to these other tables. To add a column from one of these parent tables to the form, you may click on the foreign key column name in the SQL column list, and the columns in the parent table will be available. Therefore, by linking only the lowest table in the rank, all tables in the hierarchical scheme can be referenced through the lowest ranking table.

Lookups

Lookups

What is a Lookup?

A Lookup is a form designed for listing all available records (or rows) in a table for selection and use on another form (the Profile form, a Library Maintenance form, etc.). Lookup Buttons can be attached to Edit Fields on forms so that users may associate validated information in these fields. Lookups are displayed on forms by an ellipsis button attached to an Edit field.

New Table Lookups

After you have saved a new table and clicked on OK to create a Lookup for the new table, a screen similar to the following will appear. By default, all columns that the Administrator created in the new table will appear on the Lookup form.

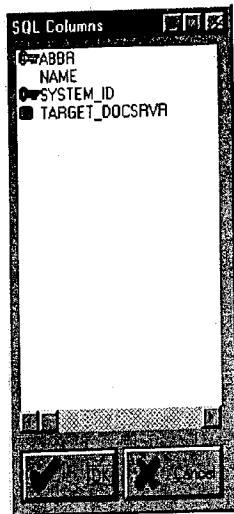
Abbreviation	Name
--------------	------

Add Column Delete Column OK Cancel

File Description: Table Lookup for BLDG

Add a Column to the Lookup

To add another column to the Lookup form for this table, click on the Add Column button. A list of available columns for that table will appear, as shown below:



Click once on each column you wish to add to the Lookup form, and then click on the OK button to return to the Lookup form. The newly added columns will then be displayed.

Changing Lookup Columns

Sort

You may change the order of the listed columns by clicking on the column's heading (the prompt for the column) and dragging it to where you would like it placed. The cursor will change from a pointer to a directional symbol until you drop the column.

Size

You may change the width of each column by clicking on the edge of the column's title. The cursor will change from a pointer to a cross hair with arrows pointing to the left and right. Drag the column's edge to the desired width, and then release the mouse button.

Title

The default title for each column is the text stored in the column's Prompt field. To change the title for the column, choose a title by first clicking once on the title, and then click on the Title button. A screen like the following will appear:

Rules for Filters

Using the filter example given above, follow the rules below for setting up the Filter statement:

1. The statement *without* the percent sign (%) must be a column present on the *lookup* form.
2. The statement *with* the percent sign must be a column represented on the *profile* form.

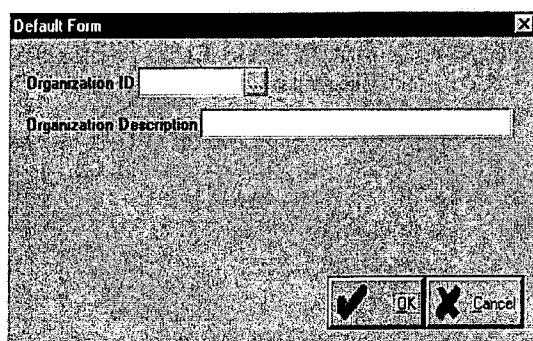
Save the Table Lookup

When you have completed making changes for the Lookup, click once on the OK button. You will then be asked if you wish to link the new table to the PROFILE table. Follow the procedures outlined below.

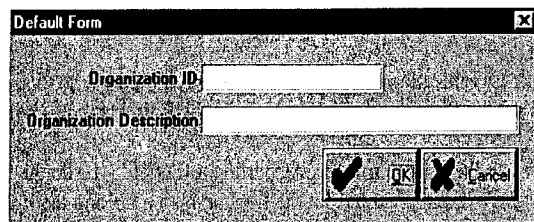
Create a Maintenance Form for the ORG Table

To create a Maintenance form for the ORG table:

1. From the main DOCS Designer screen, click the Maintenance button. A listing of tables appears.
2. Highlight ORG and click OK. A default form appears with only the OK, Cancel, and Close buttons (Cancel and Close overlap).
3. From the SQL Columns list, drag ORG_ID and ORG_NAME on to the maintenance form. Your form will look similar to the following:



4. The ORG_ID column has a lookup. Double-click the field to display the Edit Attributes window. Delete the lookup specified in the Lookup Name field. Click OK to return to the maintenance form.
5. Move and align the fields and push buttons on the form. Resize the form to encompass just the fields and push buttons.



6. Choose Save from the File menu. For the form description, enter "Organization Maintenance."
7. To exit the ORG maintenance form, choose Close from the File menu.

Exhibit Z-25

STEPHEN J. ZIZZI

SUMMARY

Mr. Zizzi is a highly inquisitive, creative and resourceful leader with 15 years experience in business and fiscal planning. He exhibits excellent skills in communication, collaboration and delegation with a comprehensive knowledge of information systems, manufacturing and production automation, software architecture and application development. Mr. Zizzi specializes in the design and development of web enabled N tier solutions and mobile wireless systems that are aligned with our client's business requirements.

TECHNICAL SKILLS

OPERATING SYSTEMS

Windows 95, 98, NT-4 Server, 2000 Adv. Server, HP-UX, Solaris, and Linux

DEVELOPMENT TOOLS

Visual Basic 5.0-6.0, OLE, MS ASP, Visual C++, Java, JavaScript, VB Script, HTML, Oracle Web DB 2.1, Oracle iPortals

RDBMS TOOLS

MS SQL 7, SQL 2000, Oracle 8i, Sybase, MS Access 9x - 2000

EXPERIENCE

CMA Consulting Services

Latham, New York

Vice President of Engineering &

Chief Software Architect

June 2000 - Present

- Designed mobile/wireless system architecture for multiple applications for the NYC Department of Citywide Administrative Services and NYS Office of General Services.
- Designed security models/systems for web and client/server based solutions.
- Developed web solution methodologies for e-Commerce, e-Fulfillment, Internet, Intranet, and Extranet systems.
- Reviewed current web development technologies and develop a corporate N tier strategy.
- Designed web based architecture and business objects for CMA's Human Resource Information System and Electronic Timesheet products.
- Technical mentor to CMA consultants - assisting in resolving technical issues with development tools and environments.
- Technical architect/lead with the MWBE project at the NYS Dormitory Authority. Engaged in data conversions from mainframe and external systems, and interfaces to MS SQL 7 and COM component design.
-

Keane, Inc

Albany, New York

Sr. Programmer/Analyst

February 1999 – May 2000

- Member of the development team of "RealNet," an Intranet system developed for information management for the NYS Office of General Services. Primary responsibilities included :

- Designed and programmed Active-X controls for web based information reporting
- Designed and programmed active server pages, Microsoft Transaction Server, and Visual Basic 6.0 components. Completed coding tasks using OLE.
- Extensive use of Microsoft SQL 6.5 T-SQL programming for report queries.

Maz Technologies, Inc

Irvine, California

Co-founder and Chief Technology Officer

April 1996 – December 1998

- Designed and managed development of a complete line of Windows information security products. Core technologies of the “IntelliGard” product line included file and intra-file encryption, smart card key management, digital authentication and biometric authentication end user applications and developer SDKs.
- Authored user interface, prototype, and proof of concept applications.
- Managed an in house development team and outsource contracting.
- Seamless integration of security technology into medium to large scale third party EDM (electronic document management) systems. Involved extensive use of event handling, windows messaging and callbacks, OLE, Visual Basic 5.0, SQL interface to Microsoft SQL6.5, Oracle RDMS, Sybase and Informix databases.
- Designed innovative, state of the art applications of smart card and biometrics technologies.
- Developed secure web applications and programmer SDK's for ISO 7816 smart card authentication embedding in Intranet and Extranet systems.
- Utilized MS SQL for dynamic content development and security maintenance.

Litronic Industries, Inc.

Irvine, California

Director of Marketing

February 1995 – April 1996

- Litronic is a leading military and government supplier of information security and classified network technology. Joined Litronic to oversee government programs, analyze and develop commercial applications of core technologies, and position the company's marketing and sales for an IPO (initial public offering).
- Developed marketing and fiscal plans for public offering memorandum.
- Designed and managed corporate Internet and Intranet sites.
- Designed, managed and introduced 3 new “commercial grade” security products from military technology.
- Developed Public key management systems (key rings) utilizing both Watcom and MS SQL databases.
- Managed development contract with Netscape and Microsoft to integrate smart card technology into web browsers for remote access and SET (secure electronic transaction).

Orbotech, Inc.

Santa Ana, California

Area Sales Manager - South Western US.

June 1992 – February 1995

- Orbotech is an Israeli based worldwide leader of manufacturing automation equipment for the electronic industry, with the U.S. headquarters in Boston, MA. The typical sale was a complex integration of Orbotech CAD/CAM systems, post processors of CAM data to N/C manufacturing systems, and implementation of Orbotech's AOI (automatic optical inspection) finished article machine vision inspection system.
- Developed management/automation reporting system based on Oracle RDMS for Orbotech's CAD/CAM systems.
- Region achieved 80% AOI market share and 65% CAD/CAM market share.
- Region consistently achieved 120% of fiscal goals.
- Participated in worldwide engineering and planning for product strategies and development.
- Managed a 25-user office network.

MacDonnell Dettwiler, Inc.

Anaheim, California

Director of U.S. Sales and Marketing

June 1988 – May 1992

- MacDonnell Dettwiler is the largest high technology contractor in Canada. The company employed over 400 scientists and engineers to develop airborne SAR (synthetic aperture radar) systems, satellite ground stations, high resolution photo imaging systems, and photo-tooling systems for electronic manufacturing. Hired with 4 others to open a U.S. headquarters to market technologies and devices developed under contract to commercial and government applications. Application areas included electronic designers and manufacturers, motion picture animation imaging, satellite reconnaissance, and satellite imaging ground systems.
- Successfully open and staffed U.S. office in the Los Angeles and Boston areas.
- Achieved \$5.7 million in the first year of operation.
- Consistently achieved 125% of fiscal goals.
- Opened new markets with film imaging for Pixar (Lucas Film) and Digital Labs.
- Designed and implemented a customer service data system and field engineer staff.

Calay Systems, Inc.

Irvine, CA

Director of Marketing and Product Development September 1983 – April 1988

- Calay Systems was the leading supplier of DA (Design Automation) and CAD systems for electronic circuit design. As a software developer and system integrator, Calay hosted systems on DEC PDP-11 hardware, with custom add in hardware accelerators. Brought in to migrate to new platforms and remove hardware dependencies, increase the company's visibility in the market and to the investment community, and to position the company for a merger or acquisition in a highly competitive period in the electronics design and manufacturing market.
- Migrated the company's products to DEC VAX and Sun 2 and 3 UNIX systems. Specified third party graphics and I/O subsystems.
- Designed and repackaged the user interface and software accelerators.

- Worked with Marketing Communications staff to increase visibility through trade shows, press tours, and public relations placements.
- The company was successfully acquired in 1987 by a German conglomerate.

EDUCATION

Cornell University

Ithaca, New York

Master in Business Administration (MBA)

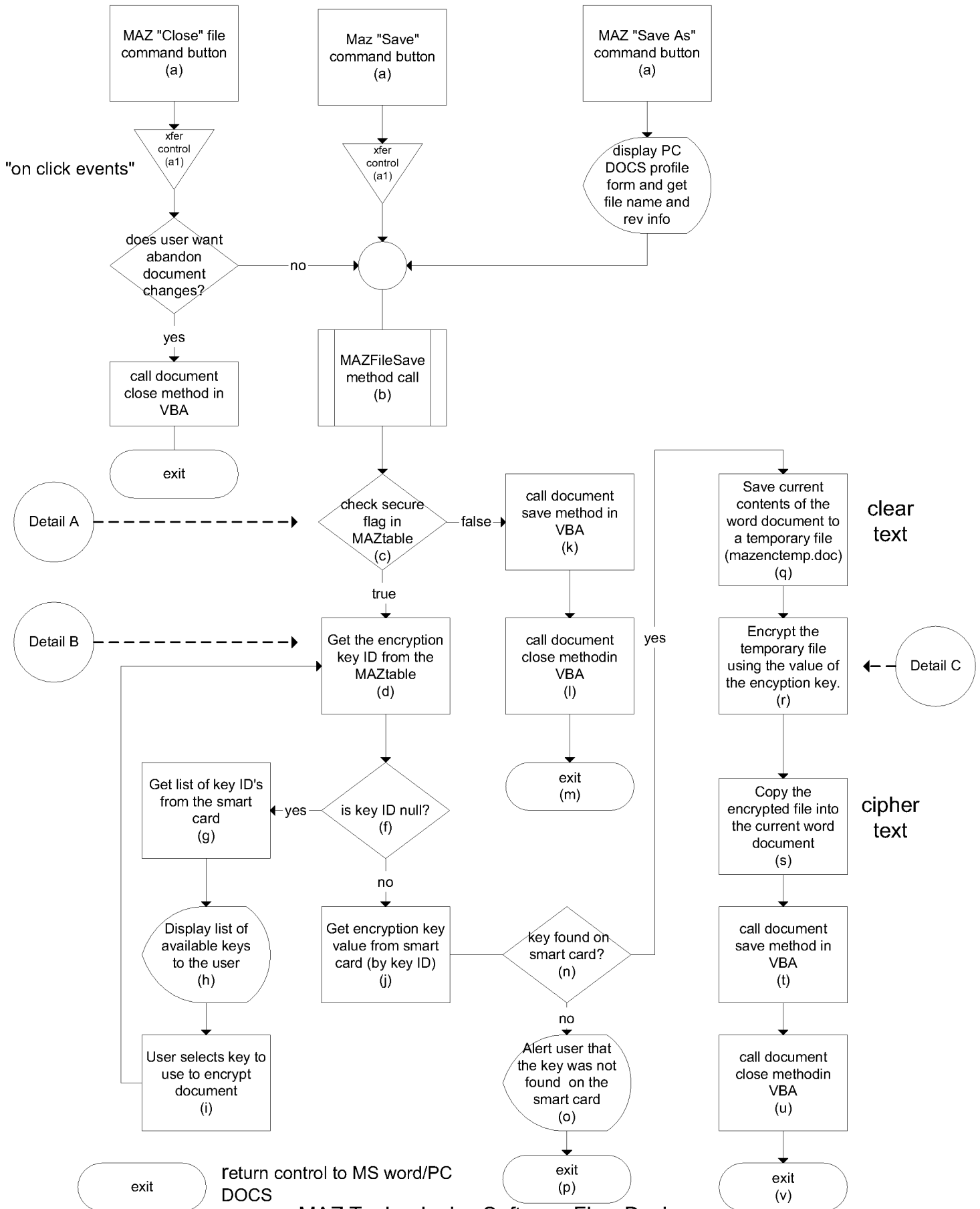
Bachelor of Science in Electrical Engineering (BS)

SPECIALIZED TRAINING

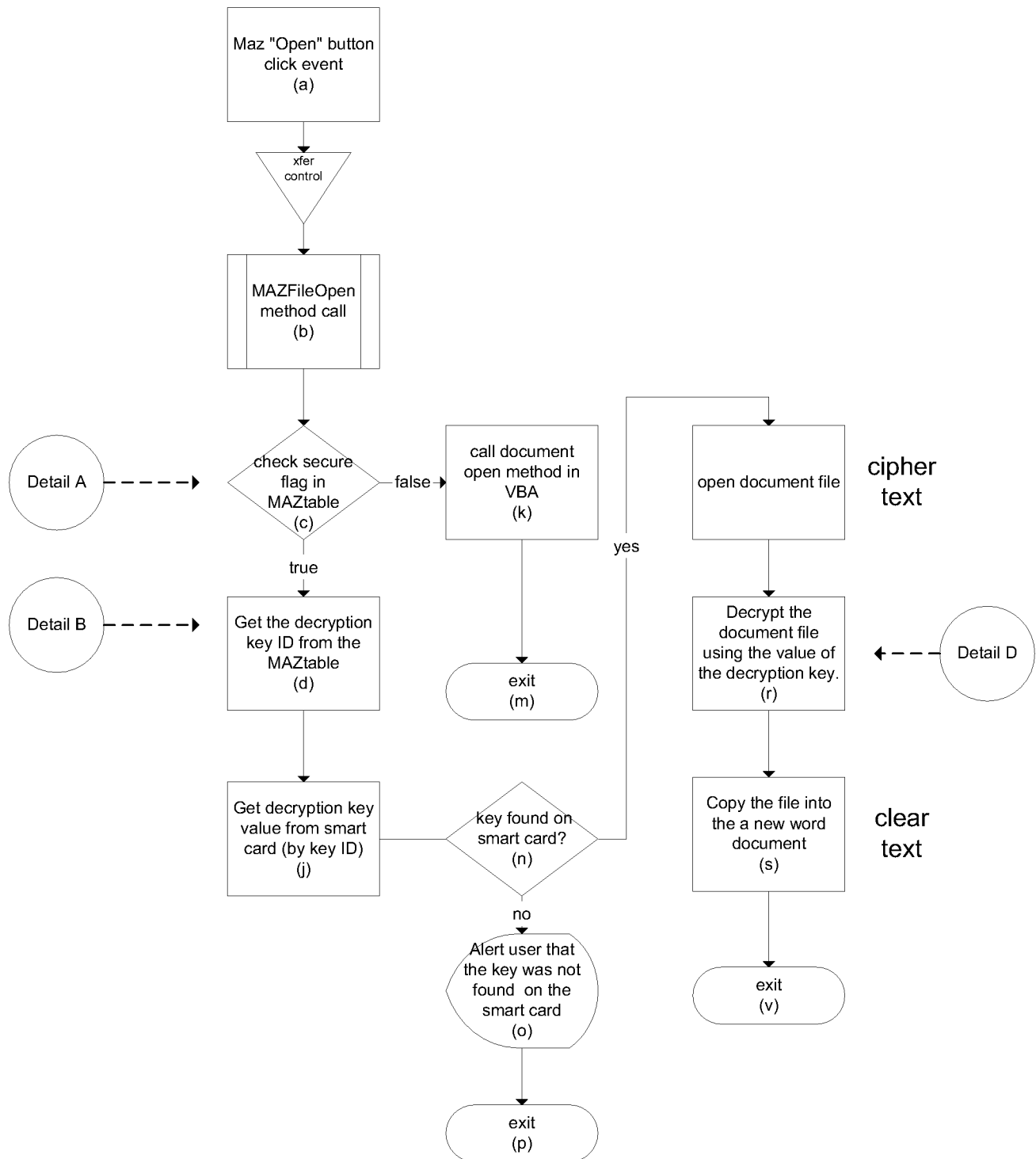
Microsoft Solution Provider, Microsoft SQL, Novell System Administrator, HP-UX, Sun OS.

EXHIBIT Z-27

Document "Save" Program Flow Chart (encrypt)

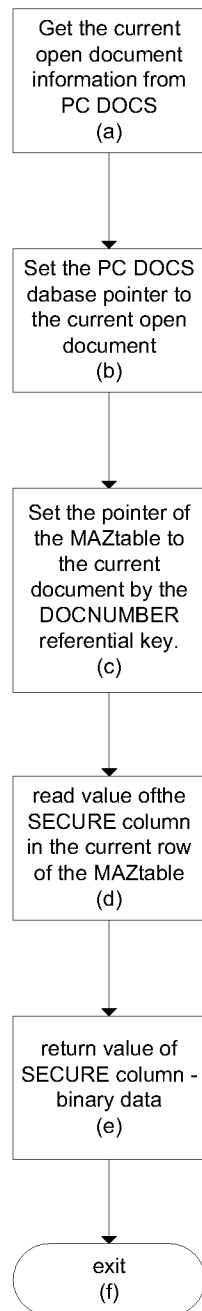


Document "Open" Program Flow Chart (decrypt)



exit
 return control to MS word/PC
DOCS

Detail A - Deturmining if a file should be encrypted



notes

[Use the DOCS Open SQLview interface "DOCSBEOpenDocument" to get the current open document]

[Use the DOCS Open SQLview interface "DOCSBESelectDocument" to set the database pointer to the proper row]

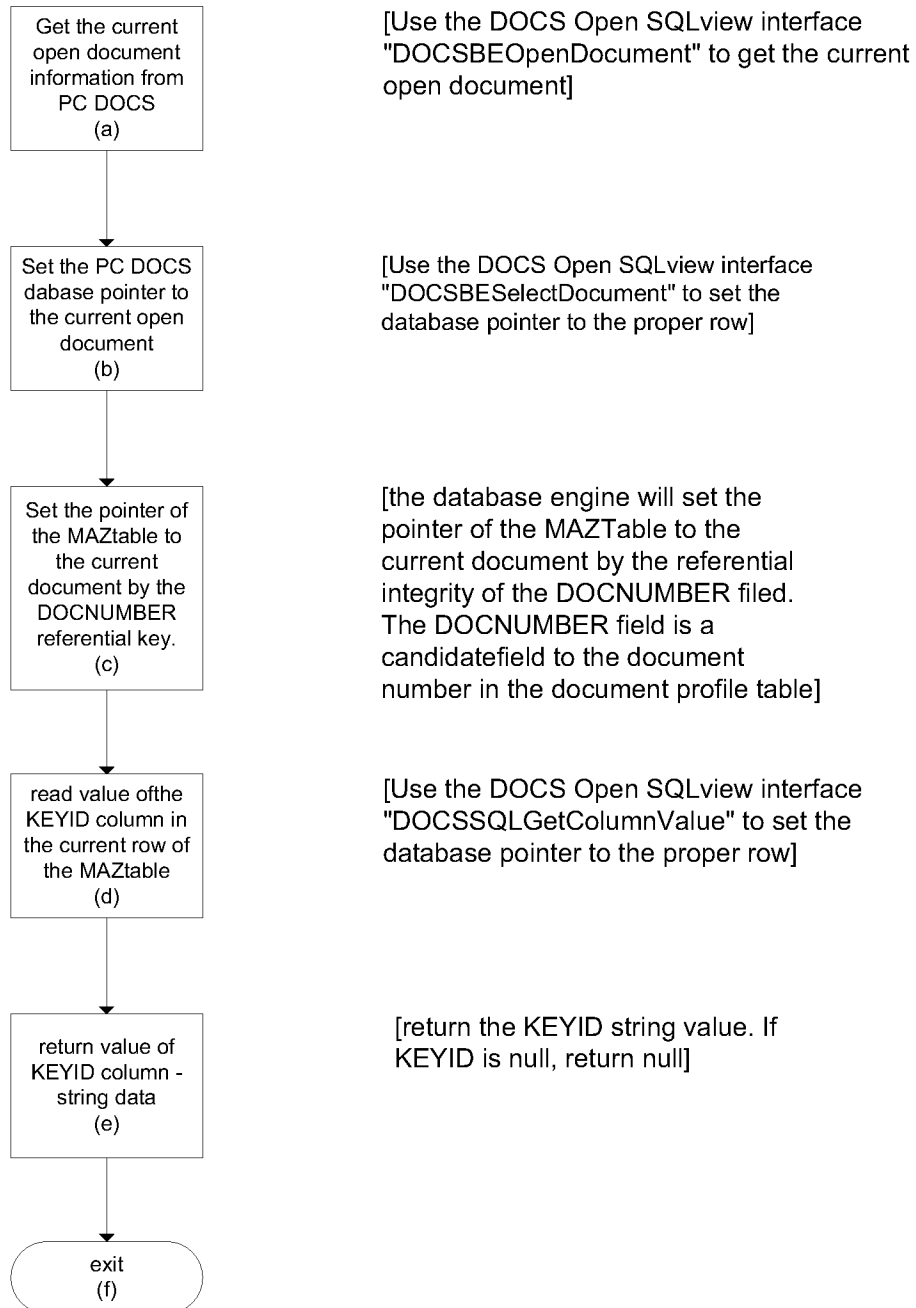
[the database engine will set the pointer of the MAZTable to the current document by the referential integrity of the DOCNUMBER filed. The DOCNUMBER field is a candidatefield to the document number in the document profile table]

[Use the DOCS Open SQLview interface "DOCSSQLGetColumnValue" to set the database pointer to the proper row]

[return bit value of SECURE. 1(true) indicates document should be encrypted]

Detail B - Fetching the value of the KeyID associated to a document

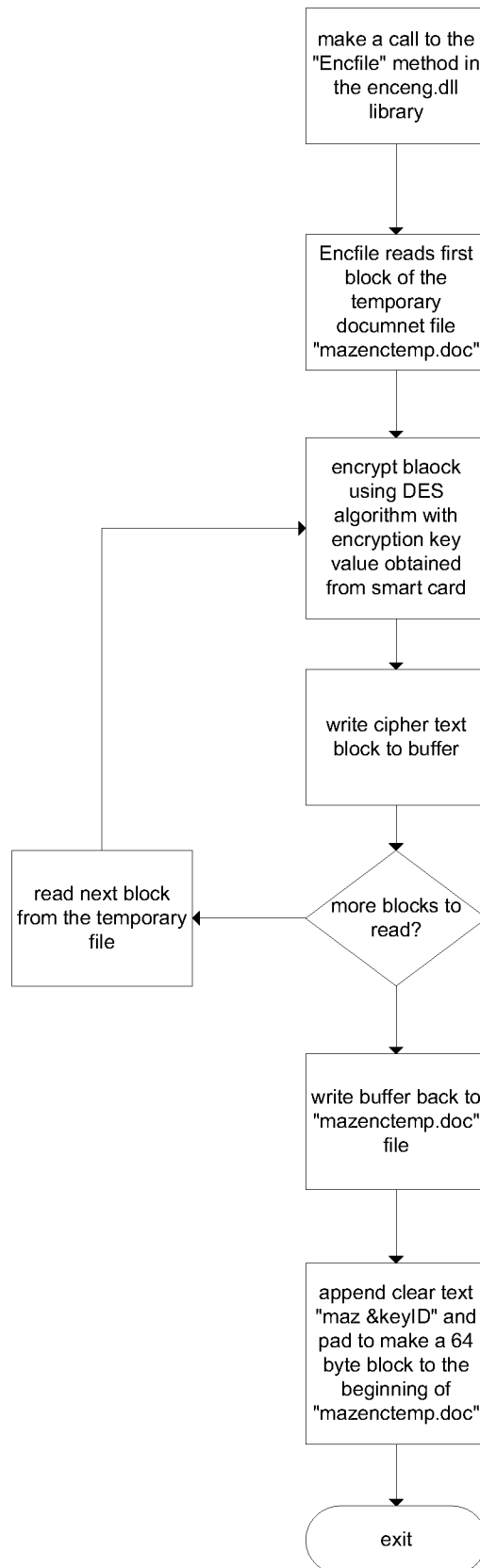
notes - do NOT assume pointers are set from last read of MAZTable!



Detail C - File encryption using DES CBC method

notes:

[the enceng.dll library is installed with
the IntelliGard main console]



[DES algorithm obtained from the
public domain standard]

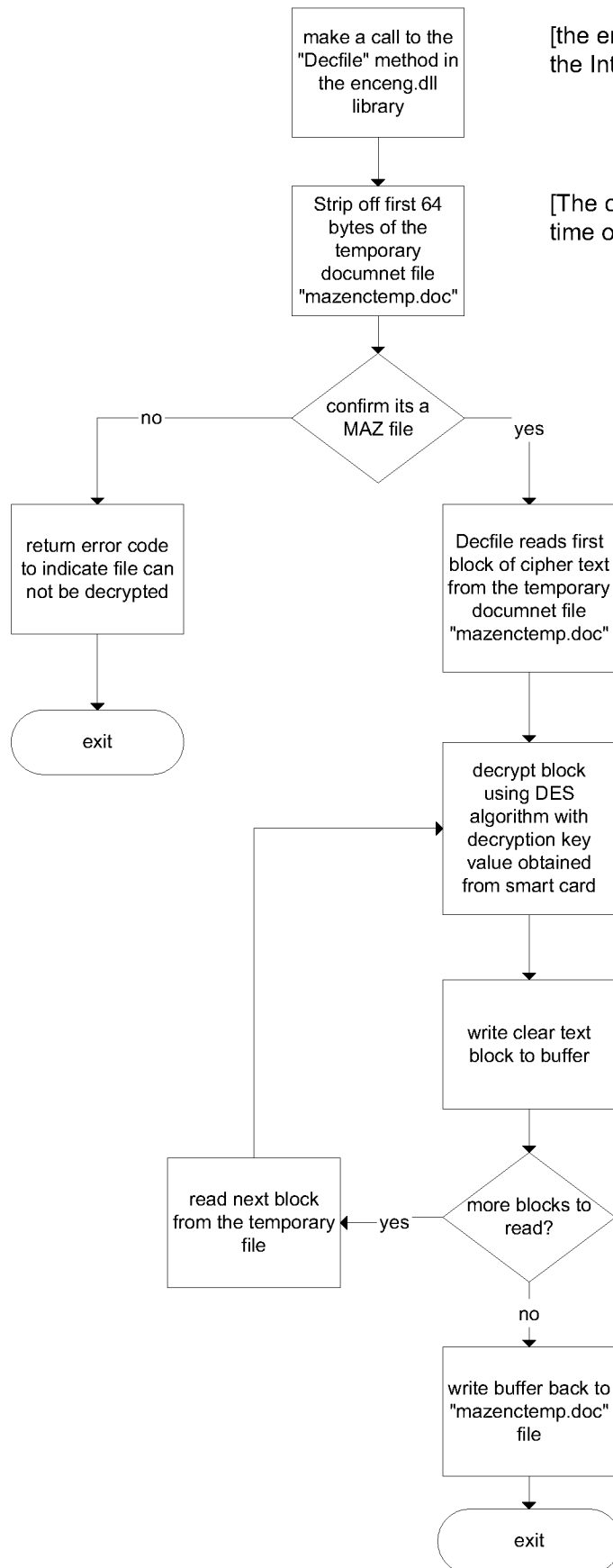
[The clear text "header" will be stored with
the file and stripped of at decryption time]

Detail D - File decryption using DES CBC method

notes:

[the enceng.dll library is installed with
the IntelliGard main console]

[The clear text "header" was added at the
time of encryption]



[DES algorithm obtained from the
public domain standard]

EXHIBIT Z-28



New Clients

Billiton plc, based in London, England, the second largest mining company in the world, has chosen DOCS Enterprise Suite Open as its document management system because of its versatility, ease of use and its ability to work on the majority of platforms. The company has implemented DOCS Open in the UK, South Africa and Holland, with more than 30 countries planned to follow.

Vagverket, the Swedish Government agency responsible for road building has purchased a site licence for 3,000 users. The testing criteria was simple: "Install it, give us three days overview training and then leave us to evaluate your product". PC DOCS was the only document management vendor that allowed them to achieve their objective.

Ernst & Young LLP, is installing a 3,500-seat EDM solution for its US tax practice.

BankBoston Corp, which has operations in 23 countries around the world, will use DOCS Open to automate its electronic customer file process.

Mercedes Benz of Stuttgart, Germany will implement DOCS Open as its EDM solution of choice and plans to roll it out to 10,000 users over the next three years.

Weil, Gotshal & Manges LLP, an international law firm based in New York, signed an agreement for CMS OPEN. The 600-lawyer firm is ranked fifth largest in the United States.

Coopers & Lybrand L.L.P. selected CMS OPEN's Mobile Time & Expenses Entry product for use by 16,000 professionals in the United States and Mexico.

MetLife's legal department licensed CompInfo's LawPack to eliminate five legacy systems with one integrated matter management solution. With over 300 in-house attorneys working nation-wide, they had a critical business problem of sharing and managing important information.

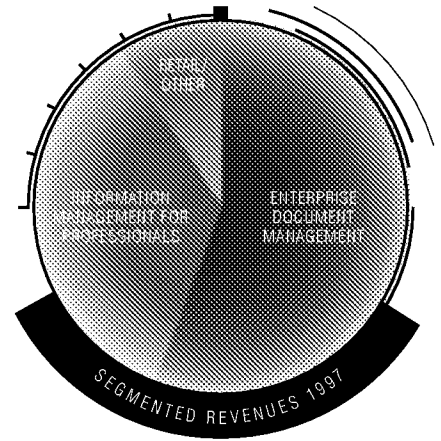
Other Noteworthy Developments

PC DOCS Enterprise Suite received the blue ribbon award from *Network World* magazine in a competitive review of document management systems.

Integrated Computer Management, a PC DOCS premier integrator, won the "Best Business Solution" in Microsoft's Solution Provider Awards competition for an EDM and imaging system based on DOCS Open and Windows NT for Ernst & Young LLP's 3,500 tax professionals.

EDS, Digital Equipment Corp. and Microsoft co-hosted a technical briefing in Washington, D.C. on PC DOCS exclusively for government professionals.

Microsoft sponsored a series of seminars in major U.S. cities to demonstrate the power of combining DOCS Enterprise Suite and Microsoft Windows NT Server, Microsoft SQL Server and Microsoft Office 97.



The Future

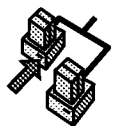
We have experienced tremendous growth over the last three years with revenues increasing from \$29 million in 1994 to \$96 million in 1997. This pace of growth, coupled with the future expansion opportunities in all our markets and an increasingly competitive landscape, required continued investment in all areas of the Company. These included research and development for new products and product maintenance; support and services to maintain the existing large customer base as well as new customers; sales and marketing to capitalize on opportunities in the market place; and administration to build the infrastructure required to support the Company's growth to the next tier. While in 1997 we continued to build the organization, our revenue growth, although significant at 18%, was not at the level we had anticipated, and resulted in lower than expected earnings for the year. The reasons for the shortfall can be attributed to a number of factors – both internal and external. We are examining these factors and are currently implementing plans to address the market issues and to balance expenses to revenue growth.

Successful high technology companies must continually evolve in order to keep pace in a rapidly changing industry. We are constantly re-evaluating our strategies, tactics and our direction and are making the changes required to maintain our leadership position in all our markets. In addition, we are aggressively investing in our products to ensure that we are continually supporting the newest technologies, including the Internet. We expect that this process will produce a stronger and more focused Company and will position us for continued revenue growth and increased profitability.

In Conclusion

I would like to extend my sincerest appreciation to our employees, customers, partners, the Board of Directors and fellow shareholders – all of whose loyalty, support and enthusiasm have contributed to our continuing success.

Rubin I. Osten
Chairman, President and Chief Executive Officer



AUDITORS' REPORT - U.S. GAAP

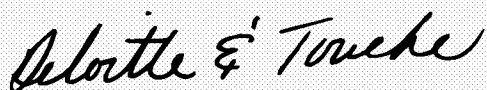
THE BOARD OF DIRECTORS AND SHAREHOLDERS OF PC DOCS GROUP INTERNATIONAL INC.

We have audited the accompanying consolidated balance sheets of PC DOCS Group International Inc. and subsidiaries as of June 30, 1997 and 1996 and the related consolidated statements of income, shareholders' equity and cash flows for each of the three years in the period ended June 30, 1997. These financial statements are the responsibility of the Company's management. Our responsibility is to express an opinion on these financial statements based on our audits. The consolidated financial statements give retroactive effect to the merger of PC DOCS Group International Inc. and DataRamp, Inc. which has been accounted for as a pooling of interests as described in Note 10 to the consolidated financial statements.

We conducted our audits in accordance with generally accepted auditing standards in the United States. Those standards require that we plan and perform the audit to obtain reasonable assurance about whether the financial statements are free of material misstatement. An audit includes examining, on a test basis, evidence supporting the amounts and disclosures in the financial statements. An audit also includes assessing the accounting principles used and significant estimates made by management, as well as evaluating the overall financial statement presentation. We believe that our audits provide a reasonable basis for our opinion.

In our opinion, these consolidated financial statements present fairly, in all material respects, the financial position of the Company as of June 30, 1997 and 1996 and the results of their operations and their cash flows for each of the three years in the period ended June 30, 1997, in conformity with generally accepted accounting principles in the United States.

We have also reported on the financial statements of PC DOCS Group International Inc. for the same period presented in accordance with accounting principles generally accepted in Canada. The significant differences between the accounting principles generally accepted in the United States and in Canada are summarized in Note 14 to those statements.



Toronto, Ontario
Deloitte & Touche
July 29, 1997

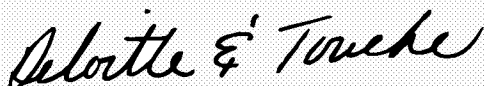
AUDITORS' REPORT - CANADIAN GAAP

TO THE SHAREHOLDERS OF PC DOCS GROUP INTERNATIONAL INC.

We have audited the consolidated statements of financial position of PC DOCS Group International Inc. as at June 30, 1997 and 1996 and the consolidated statements of earnings, retained earnings and changes in financial position for each of the years in the three year period ended June 30, 1997. These financial statements are the responsibility of the Company's management. Our responsibility is to express an opinion on these financial statements based on our audits.

We conducted our audits in accordance with auditing standards generally accepted in Canada. Those standards require that we plan and perform an audit to obtain reasonable assurance whether the financial statements are free of material misstatements. An audit includes examining, on a test basis, evidence supporting the amounts and disclosures in the financial statements. An audit also includes assessing the accounting principles used and significant estimates made by management, as well as evaluating the overall financial statement presentation.

In our opinion, these consolidated financial statements present fairly, in all material respects, the financial position of the Company as at June 30, 1997 and 1996 and the results of its operations and the changes in its financial position for each of the years in the three year period ended June 30, 1997, in accordance with accounting principles generally accepted in Canada.



Toronto, Ontario
Deloitte & Touche
July 29, 1997

EXHIBIT Z-29

Option Explicit

' Key Server Class - December 1997, SJZ

Public Property Get NullKeyValue(ByVal sName As String) As String ' Used by iWatch or other C calling program
 DRK = False

If sCardType <> "S" Then

SCRet = CLX_CardInserted

If SCRet = 0 Then

MsgRet = MsgBox("Please Insert Your Smart Card and Hit OK", vbExclamation + vbOKOnly, "IntelliGard Smart Suite")
 End If

If CheckCardID Then

NullKeyValue = ReadKeyValue(sName) & Chr(0)
 End If

Else

If sSoftConfirm = "Y" Then

'/* If password confirm is set

frmSoftConfirm.Show vbModal

If LoginSucceeded = False Then

NullKeyValue = "empty"

Else

NullKeyValue = ReadSoftKeyValue(sName) & Chr(0)

End If

End If

End If

DRK = False

End Property

the key name or key ID

Public Property Get KeyValue(sName As String) As String
 DRK = False

' Used by Vbasic Calling Programs

DRK = False

If sCardType <> "S" Then

[this checks for a file based, non-smart card use]

SCRet = CLX_CardInserted *- this is a call to the scrcader.dll (modAPI pg.1). The dll is the device interface to the manufacturers reader device*

If SCRet = 0 Then

SCRet = MsgBox("Please Insert Your Smart Card and Hit OK", vbExclamation + vbOKOnly, "IntelliGard Smart Suite")
 End If

If CheckCardID Then

KeyValue = ReadKeyValue(sName) *branch to modCard API pg 7*

End If

sets the property on exit.

Else

If sSoftConfirm = "Y" Then

'/* If password confirm is set

frmSoftConfirm.Show vbModal

If LoginSucceeded = False Then

KeyValue = "empty"

Else

KeyValue = ReadSoftKeyValue(sName)

End If

End If

End If

DRK = False

End Property


```
Public Function ChangePassword(sOldPassword As String, sNewpassword As String) As Boolean
Dim NewPass As String
Dim OldPass As String
Dim CardPass As String

OldPass = Trim(sOldPassword)
NewPass = Trim(sNewpassword)

If sCardType <> "S" Then

    CardPass = sCardPassword()

    SCRet = CLX_CardInserted

    If SCRet = 0 Then
        SCRet = MsgBox("Please make sure your smart card is inserted and Hit OK", vbExclamation + vbOK
Only, "IntelliGard Smart Suite")
    End If

    If CheckCardID Then
        If Trim(UCase(OldPass)) <> Trim(UCase(CardPass)) Then
            RetOK = MsgBox("Unable to Authorize Password Change, Hit OK to Continue", vbExclamation
+ vbOKOnly, "IntelliGard Smart Suite")
            ChangePassword = False
        Else
            SetPassword NewPass
            ChangePassword = True
        End If
    End If

Else

    If Trim(UCase(OldPass)) <> Trim(UCase(sPassword)) Then
        RetOK = MsgBox("Unable to Authorize Password Change, Hit OK to Continue", vbExclamation
+ vbOKOnly, "IntelliGard Smart Suite")
        ChangePassword = False
    Else
        NewPass = Trim(sNewpassword)

        If OpenKeyFile Then
            Sleep (1000)
            CloseKeyFile (NewPass)
            ChangePassword = True
            sPassword = NewPass
        End If
    End If

End If

End Function

Public Property Get KNames() As Collection
Dim cKeys As New Collection
Dim n As Integer

ReaderBusy = True

For n = 1 To Val(NumKeys)

    cKeys.Add kMtxName(n)

Next n

Set KNames = cKeys

ReaderBusy = False

End Property
```

```
Public Property Get KeyNames() As Collection
    Dim cKeys As New Collection
    Dim n As Integer
```

```
    ReaderBusy = True
```

```
    For n = 1 To Val(NumKeys)
```

```
        cKeys.Add kMtxName(n)
```

```
    Next n
```

```
    Set KeyNames = cKeys
```

```
    ReaderBusy = False
```

```
End Property
```

```
Public Function DisplayReturnKey(dkeyname As String, dKeyValue As String) As Boolean
```

```
    DRK = True
```

```
    bECB = False
```

```
    If sCardType <> "S" Then
```

```
        SCRet = CLX_CardInserted
```

```
        If SCRet = 0 Then
```

```
            SCRet = MsgBox("Please Insert Your Smart Card and Hit OK", vbExclamation + vbOKOnly, "IntelliG  
ard Smart Suite")
```

```
        End If
```

```
        If CheckCardID Then
```

```
            frmSelectKey.Show vbModal
```

```
            dkeyname = scKeyName
```

```
            dKeyValue = scKeyValue
```

```
            DisplayReturnKey = True
```

```
        End If
```

```
    Else
```

```
        If sSoftConfirm = "Y" Then
```

```
            frmSoftConfirm.Show vbModal
```

```
            If LoginSucceeded = False Then
```

```
                dkeyname = "cancel"
```

```
                dKeyValue = "empty"
```

```
                DisplayReturnKey = False
```

```
                Exit Function
```

```
            End If
```

```
        End If
```

```
        frmSelectKey.Show vbModal
```

```
        dkeyname = scKeyName
```

```
        dKeyValue = scKeyValue
```

```
        DisplayReturnKey = True
```

```
    End If
```

```
End Function
```

```
Public Function NullDisplayReturnKey(dkeyname As String, dKeyValue As String) As Boolean ' Used b  
y iWatch or other C calling program
```

```
    DRK = True
```

```
    bECB = False
```

```
    If sCardType <> "S" Then
```

```
        SCRet = CLX_CardInserted
```

```
        '/* Procedure for soft keys  
        '/* If password confirm is set
```

```
If SCRet = 0 Then
    SCRet = MsgBox("Please Insert Your Smart Card and Hit OK", vbExclamation + vbOKOnly, "Intelli
Gard Smart Suite")
End If

If CheckCardID Then
    frmSelectKey.Show vbModal
    dkeyname = scKeyName & Chr(0)
    dKeyValue = scKeyValue & Chr(0)
    NullDisplayReturnKey = True
End If

Else
    If sSoftConfirm = "Y" Then
        frmSoftConfirm.Show vbModal
        If LoginSucceeded = False Then
            dkeyname = "cancel" & Chr(0)
            dKeyValue = "empty" & Chr(0)
            NullDisplayReturnKey = False
            Exit Function
        End If
    End If

    frmSelectKey.Show vbModal
    dkeyname = scKeyName & Chr(0)
    dKeyValue = scKeyValue & Chr(0)
    NullDisplayReturnKey = True
End If

End Function

Public Function DisplayKeys() As Boolean

SCRet = CLX_CardInserted

If SCRet = 0 Then
    SCRet = MsgBox("Please Insert Your Smart Card and Hit OK", vbExclamation + vbOKOnly, "IntelliGar
d Smart Suite")
End If

    frmSelectKey.Show vbModal
    DisplayKeys = True

End Function

Public Function KillCard() As Boolean
KillCard = True
End Function

Public Function ValidRequest(code As String) As Boolean

End Function
```

Option Explicit

' Reader Calls

Screader.dll was a smart card reader device interface

```
Public Declare Function CLX_OpenReader Lib "screader.dll" (ByVal port As Integer, ByVal baud As Long) As Integer
Public Declare Function CLX_CloseReader Lib "screader.dll" () As Integer
Public Declare Function CLX_ResetReader Lib "screader.dll" () As Integer
Public Declare Function CLX_GreenLedOn Lib "screader.dll" () As Integer
Public Declare Function CLX_GreenLedOff Lib "screader.dll" () As Integer
Public Declare Function CLX_RedLedOn Lib "screader.dll" () As Integer
Public Declare Function CLX_RedLedOff Lib "screader.dll" () As Integer
Public Declare Function CLX_SetCardType Lib "screader.dll" (ByVal X As Integer) As Integer
Public Declare Function CLX_CardInserted Lib "screader.dll" () As Integer
```

' Async Processor Cards

```
Private Declare Function CLX_CardOn Lib "screader.dll" (ByVal rdata As Any) As Integer
Private Declare Function CLX_CardOff Lib "screader.dll" () As Integer
Private Declare Function CLX_Invalidate7816 Lib "screader.dll" () As Integer
Private Declare Function CLX_Rehab7816 Lib "screader.dll" () As Integer
Private Declare Function CLX_WriteBinary7816 Lib "screader.dll" (ByVal Address As Long, ByVal DataLen As Long, ByVal DataBuf As Any) As Integer
Private Declare Function CLX_ReadBinary7816 Lib "screader.dll" (ByVal Address As Long, ByVal DataLen As Long, ByVal DataBuf As Any) As Integer
Private Declare Function CLX_UpdateBinary7816 Lib "screader.dll" (ByVal Address As Integer, ByVal P3 As Long, ByVal DataBuf As Any) As Integer
Private Declare Function CLX_Verify7816 Lib "screader.dll" (ByVal DataLen As Long, ByVal DataBuf As Any) As Integer
Private Declare Function CLX_Select7816 Lib "screader.dll" (ByVal DataLen As Long, ByVal DataBuf As Any) As Integer
Private Declare Function CLX_Change_Code7816 Lib "screader.dll" (ByVal DataLen As Long, ByVal DataBuf As Any) As Integer
```

Public Sub BootCardServer()

BootServer

```
CurrentCardID = ReadCard(304, 7)
sUserType = Mid(CurrentCardID, 1, 1)
NumKeys = Mid(CurrentCardID, 2, 1)
```

```
If sUserType = "U" Then
    ' frmSplash.disdesktop.DisableTaskSwitching = True
Else
    ' frmSplash.disdesktop.DisableTaskSwitching = False
End If
```

GetKeyNames

frmSplash.tmrCardPoll.Enabled = True

End Sub

Public Function bOpenReader() As Boolean ' Sets SessionOpen (Reader Open)

```
SCRet = 0
Select Case sCardReaderType

    Case "A"
        nReaderPort = Val(nReaderPort - 1)
        SCRet = CLX_OpenReader(nReaderPort, 9600)

    Case "B"
        SCRet = CLX_OpenReader(0, 0)
```

End Select

Select Case SCRet

```
Case 0
    bOpenReader = True
```

```
ReaderOpen = True
SCRet = CLX_GreenLedOn
```

Case 1

```
frmSplash.lslError.Visible = True
frmSplash.lslError.AddItem " PROBLEM OPENING SERIAL PORT"
frmSplash.lslError.AddItem " (Error 501) "
frmSplash.lslError.AddItem " "
frmSplash.lslError.AddItem "TROUBLE SHOOTING HELP:"
frmSplash.lslError.AddItem "1. Have you recently installed new hardware?"
frmSplash.lslError.AddItem "2. Have you recently reconfigured your system?"
bOpenReader = False
ReaderOpen = False
```

Case 2

```
frmSplash.lslError.Visible = True
frmSplash.lslError.AddItem "PROBLEM WITH THE CARD READER"
frmSplash.lslError.AddItem " (Error 502) "
frmSplash.lslError.AddItem " "
frmSplash.lslError.AddItem "TROUBLE SHOOTING HELP:"
frmSplash.lslError.AddItem "1. Is the Card Reader plugged in and the power on?"
frmSplash.lslError.AddItem "2. Is the cable securely connected to your computer?"
bOpenReader = False
ReaderOpen = False
End Select
```

End Function

```
Public Function bCardInserted() As Boolean ' /* Modified for both M and P type Cards and A and B readers */
```

```
Dim nReturn As Long
Dim X As Integer
Dim RetryYes As Long
X = 1
ReaderBusy = True
```

```
SCRet = CLX_CardInserted()
```

```
Select Case SCRet
```

Case 1

```
bCardInserted = True
CardOpen = True
ReaderBusy = False
X = 5
Exit Function
```

Case 0

```
BlinkRedLED
frmSplash.lslStatus.AddItem " error - smart card not detected"
frmSplash.Refresh
frmSplash.lslError.Visible = True
frmSplash.lslError.AddItem "PROBLEM WITH THE SMART CARD"
frmSplash.lslError.AddItem " (Error 503) "
frmSplash.lslError.AddItem " "
frmSplash.lslError.AddItem "TROUBLE SHOOTING HELP:"
frmSplash.lslError.AddItem "1. Is the Smart Card inserted in the reader?"
frmSplash.lslError.AddItem "2. Is it inserted with the gold contacts up?"
frmSplash.lslError.AddItem ""
frmSplash.lslError.AddItem ""
RetryYes = MsgBox("Please Insert Your Smart Card and Hit OK", vbOKCancel, "IntelliGar  
d Smart Server")
```

```
If RetryYes = 2 Then
End
End If
```

```
While X < 4
```

```
SCRet = CLX_CardInserted()
```

```
If SCRet = 1 Then
```

```
    If sCardType = "P" Then      /* Power up Card if processor card
        If bBootCard Then
            End If
        End If
```

```
        X = 5
        bCardInserted = True
        CardOpen = True
        SCRet = CLX_GreenLedOn
        ReaderBusy = False
        Exit Function
```

```
    End If
```

```
BlinkRedLED
frmSplash.lstStatus.AddItem "    error - smart card not detected"
frmSplash.Refresh
```

```
RetryYes = MsgBox("Please Insert Your Smart Card and Hit OK", vbOKCancel, "IntelliGard Smart Server")
```

```
    If RetryYes = 2 Then
        End
    End If
    X = X + 1
Wend
```

```
End Select
```

```
ReaderBusy = False
```

```
If X <> 5 Then
```

```
    SCRet = MsgBox("Could Not Locate SmartCard, Access Denied", vbCritical + vbOKOnly)
    bCardInserted = False
    CardOpen = False
```

```
End If
```

```
End Function
```

```
Public Sub BootServer()
```

```
frmSplash.Show
frmSplash.Refresh
frmSplash.lstStatus.AddItem "Opening and testing card reader"
frmSplash.Refresh
```

```
ReaderBusy = True
```

```
    If Not bOpenReader Then
        MsgRet = MsgBox("Please Check the Reader and Hit OK", vbExclamation + vbOKOnly, "IntelliGard Server")
        If Not bOpenReader Then
            MsgRet = MsgBox("There is a Problem With the Reader, Please Contact Technical Support", vbCritical + vbOKOnly, "IntelliGard Server")
        End
    End If
```

```
End If
```

```
frmSplash.lstStatus.AddItem "    - card reader OK"
frmSplash.Refresh
```

```
frmSplash.lstStatus.AddItem "Opening and testing smart card"
frmSplash.Refresh
```

```
    If Not bCardInserted Then
```

```
MsgRet = MsgBox("Unable to Detect a Valid Smart Card, Server Locking System", vbExclamation
+ vbOKOnly, "IntelliGard Smart Suite")
    SystemLocked = True
    frmLock.Show vbModal
End If

' Boot up Smart Card

If sCardType = "P" Then

    If Not bBootCard Then
        MsgRet = MsgBox("Unable to Open the Smart Card for use, Server Locking System", vbExclamatio
n + vbOKOnly, "IntelliGard Smart Suite")
        SystemLocked = True
        frmLock.Show vbModal
    End If

End If

' Open Login Form and Clear Splash Screen

frmSplash.lstStatus.AddItem "    - smart card OK"
frmSplash.Refresh

' Check for Log In Requirement

If nNetLogIn = 1 Then
    frmSplash.lstStatus.AddItem "Ready to Log In"
    frmSplash.Refresh
    If Not SessionOpen Then
        UserLogIn
    End If
Else
    frmSplash.lstStatus.AddItem "Single Log In Active, Starting Server"
    frmSplash.Refresh
End If

ReaderBusy = False

frmSplash.Hide

End Sub

Public Function bBootCard() As Boolean

    Dim sBootBuffer As String * 255
    sBootBuffer = Space(255)

    SCRet = CLX_CardOn(sBootBuffer)
    Sleep (100)

    sBootBuffer = Chr$(&H3D) + Chr$(&H0)      '/*    Select the root file - &H3D00 */
    SCRet = CLX_Select7816(2, sBootBuffer)
    Sleep (100)

    sBootBuffer = Chr$(&H3D) + Chr$(&H40)      '/*    Select the EF file - &H3D40 */
    SCRet = CLX_Select7816(2, sBootBuffer)
    Sleep (100)

    sBootBuffer = Chr$(1) + Chr$(2) + Chr$(3) + Chr$(4) + Chr$(5) + Chr$(6) + Chr$(7) + Chr$(8)
    '/* Send the password to open the EF file */
    SCRet = CLX_Verify7816(8, sBootBuffer)
    Sleep (100)

    If SCRet = 1 Or Trim(Hex(SCRet)) = "9000" Then
        bBootCard = True
    Else
        bBootCard = False
    End If

End If
```

End Function

```
Public Sub UserLogin()
    Dim X As Integer
    ReaderBusy = True
    Set frmLogin = New frmLogin
    CancelLogin = False
    X = 1

    While X < 4 ' /* Start Login With Smart Card, Allow 4 attempts

        If CancelLogin Then
            SCRet = MsgBox("Login Canceled by User, Locking System", vbCritical + vbOKOnly, "IntelliGard Smart Suite")
            Unload frmLogin
            frmShutdown.Show vbModal
            Exit Sub
        End If

        If Not SessionOpen Then
            frmLogin.Show vbModal
            X = X + 1
        Else
            X = 4
        End If

    Wend

    Unload frmLogin
    If Not SessionOpen Then
        SCRet = MsgBox("Unable to Login User, Locking System", vbCritical + vbOKOnly, "IntelliGard Smart Suite")
        frmShutdown.Show vbModal
    Else
        CurrentCardID = ReadCard(304, 8)
    End If
End Sub
```

ReaderBusy = False

End Sub

```
Function sCardUserName() As String
    Dim sText As String * 16
    sText = Space(16)

    sText = ReadCard(0, 16)

    If sText <> "" Then
        sCardUserName = sTrimNulls(sText)
    Else
        sCardUserName = ""
        SCRet = MsgBox("There is a Problem with the User ID on this Card", vbExclamation + vbOKOnly, "IntelliGard Server")
    End If
    LastUserName = sTrimNulls(sText)
End Function
```

```
Function sCardPassword() As String
    Dim sText As String
    sText = Space(16)
    sText = ReadCard(16, 16)

    If sText <> "" Then
        sCardPassword = Trim(sText)
    Else
        sCardPassword = ""
        SCRet = MsgBox("There is a Problem with the user password on this Card", vbExclamation + vbOKOnly, "IntelliGard Server")
    End If
End Function
```


End Function

Public Function CheckCardID() As Boolean

Dim sGetID As String

sGetID = Space(8)

If sCardType <> "S" Then

sGetID = ReadCard(304, 8)

If CurrentCardID = sGetID Then

CheckCardID = True

Else

UserLogIn

CheckCardID = True

End If

CurrentCardID = Trim(sGetID)

Else

CurrentCardID = "U"

CheckCardID = True

End If

End Function

Public Function nCardPresent()

SCRet = CLX_CardInserted

nCardPresent = SCRet

End Function

Public Function bCloseReader() As Boolean

ReaderBusy = True

SCRet = CLX_GreenLedOff

SCRet = CLX_CloseReader

If SCRet = 0 Then

ReaderOpen = False

CardOpen = False

SessionOpen = False

LogOutEntry

bCloseReader = True

Else

bCloseReader = False

ReaderOpen = True

End If

End Function

Public Function sTrimNulls(sText As String) As String

Dim nPos As Integer

Dim nNull As Integer

Dim sNewText As String

nNull = InStr(sText, Chr\$(0))

If nNull = 0 Then

sTrimNulls = sText

Else

nPos = 1

While Asc(Mid(sText, nPos, 1)) <> 0

sNewText = sNewText & Mid(sText, nPos, 1)

nPos = nPos + 1

Wend

sTrimNulls = sNewText

End If

End Function

```
Sub SetPassword(ByVal sPassword As String)
    Dim sPass As String
    sPass = Space(16)
    SCRet = CLX_WriteBinary7816(16, 16, sPass)      ' Clear previous password
    ReaderBusy = True
    sPass = Trim(sPassword)
    SCRet = CLX_WriteBinary7816(16, Len(sPass), sPass)
    BlinkRedLED
    ReaderBusy = False
End Sub
```

```
Sub BlinkRedLED()
    If nLEDState = 1 Then
        ReaderBusy = True
        SCRet = CLX_RedLedOn
        SCRet = CLX_RedLedOff
        ReaderBusy = False
    End If
End Sub
```

```
Sub BlinkGreenLed()
    If nLEDState = 1 Then
        ReaderBusy = True
        SCRet = CLX_GreenLedOn
        SCRet = CLX_GreenLedOff
        ReaderBusy = False
    End If
End Sub
```

Public Function ReadCard(Address As Integer, nSize As Integer) As String

```
Dim TmpBuffer As String
TmpBuffer = Space(16)
```

ReaderBusy = True

```
If sCardType = "P" Then
    SCRet = CLX_ReadBinary7816(Address, nSize, TmpBuffer)
    ReadCard = Left(TmpBuffer, 14)
    BlinkRedLED
End If
```

ReaderBusy = False

End Function

```
Public Function ReadKeyValue(keyname As String) As String
    Dim nAddress As Integer
    Dim n As Integer
    Dim MName As String
    Dim XName As String
```

```
keyindex = Left(keyname, 1)
n = Asc(keyindex) - 64 ] - get a key index from name
```

```
If n > Val(NumKeys) Then
    MsgBox("The key required is out of range in the current key set. Please contact your  
system administrator", vbCritical + vbOKOnly)
    ReadKeyValue = "empty"
    Exit Function
End If
```

nAddress = (n * 24) + 24 - calculate the key value address on the smart card

```
If nAddress = 0 Then
    ' /* Test to see if key name was found & key was enable
    d */
```

```
ReadKeyValue = "empty"  
End If
```

```
MName = Trim(sTrimNulls((kMtxName(n))))  
XName = Trim(sTrimNulls(keyname))
```

```
If XName = MName Then  
ReadKeyValue = Trim(ReadCard(nAddress, 8))
```

```
Else  
ReadKeyValue = "empty"  
End If
```

```
End Function
```

```
Public Sub LoginEntry()
```

```
Dim UserLog As String  
UserLog = sWinDir & "\mazsrv.log"  
Open UserLog For Append As #1  
Print #1, "Server"; " "; LastUserName; " - logged on - "; Date; Time(); Chr$(13); Chr$(10);  
Close #1  
End Sub
```

```
Public Sub LogoutEntry()
```

```
Dim UserLog As String  
UserLog = sWinDir & "\mazsrv.log"  
Open UserLog For Append As #1  
Print #1, "Server"; " "; LastUserName; " - Logged off - "; Date; Time(); Chr$(13); Chr$(10)  
;  
Close #1  
End Sub
```

```
Public Sub GetKeyNames()
```

```
Dim n As Integer  
Dim kad As Integer  
kad = 32  
  
For n = 1 To NumKeys  
kMtxName(n) = ReadCard(kad, 16)  
kad = kad + 24  
Next n  
End Sub
```

] make sure this keyname is found on this
Smartcard. The key names matrix kMtxName
was loaded upon card insertion (see GetKeyNames
below)

If we find a valid name match, then call read card
at the address generated for the key

] loop through card addresses and
get the key names from card and
load into memory